目錄

Matplotlib 用户指南

> 来源：User's Guide
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

- 在线阅读
- PDF格式
- EPUB格式
- MOBI格式
- 代码仓库

赞助我

简介

> 原文：Introduction
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

Matplotlib 是一个用于在 Python 中绘制数组的 2D 图形库。虽然它起源于模仿 MATLAB®[1] 图形命令，但它独立于 MATLAB，可以以 Pythonic 和面向对象的方式使用。虽然 Matplotlib 主要是在纯 Python 中编写的，但它大量使用 NumPy 和其他扩展代码，即使对于大型数组也能提供良好的性能。

Matplotlib 的设计理念是，你应该能够使用几个，或者只有一个命令创建简单的图形。如果你想看到你的数据的直方图，你不需要实例化对象，调用方法，设置属性等等；它应该能够工作。

多年来，我常常使用 MATLAB 进行数据分析和可视化。 MATLAB 擅长绘制漂亮的图形。当我开始处理 EEG 数据时，我发现我需要编写应用程序来与我的数据交互，并在 MATLAB 中开发了一个 EEG 分析应用程序。随着应用程序越来越复杂，需要与数据库，http 服务器交互，并操作复杂的数据结构，我开始与 MATLAB 作为一种编程语言的限制而抗争，并决定迁移到 Python。 Python 作为一种编程语言，弥补了 MATLAB 的所有缺陷，但我很难找到一个 2D 绘图包（3D VTK 则超过了我的所有需求）。

当我去寻找一个 Python 绘图包时，我有几个要求：

- 绘图应该看起来不错 - 发布质量。 对我来说一个重要的要求是文本看起来不错（抗锯齿等）
- 用于包含 TeX 文档的 Postscript 输出
- 可嵌入图形用户界面用于应用程序开发
- 代码应该足够容易，我可以理解它，并扩展它
- 绘图应该很容易

没有找到适合我的包，我做了任何自称 Python 程序员会做的事情：撸起我的袖子开始自己造。我没有任何真正的计算机图形经验，决定模仿 MATLAB 的绘图功能，因为 MATLAB 做得很好。这有额外的优势，许多人有很多 MATLAB 的经验，因此，他们可以很快开始在 python 中绘图。从开发人员的角度来看，拥有固定的用户接口（pylab 接口）非常有用，因为代码库的内容可以重新设计，而不会影响用户代码。

Matplotlib 代码在概念上分为三个部分：pylab 接口是由 `matplotlib.pylab` 提供的函数集，允许用户使用非常类似于 MATLAB 图生成代码（Pyplot 教程）的代码创建绘图。 Matplotlib 前端或 Matplotlib API 是一组重要的类，创建和管理图形，文本，线条，图表等（艺术家教程）。这是一个对输出无所了解的抽象接口。后端是设备相关的绘图设备，也称为渲染器，将前端表示转换为打印件或显示设备（什么是后端？）。后端示例：PS 创建 PostScript® 打印件，SVG 创建可缩放矢量图

形打印件，Agg 使用 Matplotlib 附带的高质量反颗粒几何库创建 PNG 输出，GTK 在 Gtk+ 应用程序中嵌入 Matplotlib，GTKAgg 使用反颗粒渲染器创建图形并将其嵌入到 Gtk+ 应用程序中，以及用于 PDF，WxWidgets，Tkinter 等。

Matplotlib 被很多人在许多不同的上下文中使用。有些人希望自动生成 PostScript 文件以发送给打印机或发布商。其他人在 Web 应用程序服务器上部署 Matplotlib 来生成 PNG 输出，并包含在动态生成的网页中。一些人在 Windows™ 上的 Tkinter 的 Python shell 中以交互方式使用 Matplotlib。我的主要用途是将 Matplotlib 嵌入 Windows，Linux 和 Macintosh OS X 上运行的 Gtk+ EEG 应用程序中。

> [1] MATLAB 是 MathWorks 公司的注册商标。

安装

> 原文：Installing
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

有许多安装 matplotlib 的不同方法，最好的方法取决于你使用的操作系统，已经安装的内容以及如何使用它。 为了避免涉及本页上的所有细节（和潜在的复杂性），有几个方便的选项。

安装预构建包

多数平台：Python 科学分发包

第一个选项是使用已经内置 matplotlib 的预打包的 Python 分发包。`Continuum.io` Python 分发包（Anaconda 或 miniconda）和 Enthought 分发包（Canopy）都是『在 Windows，OSX 和主流 Linux 平台开箱即用并正常工作』的出色选择。 这两个分发包包括 matplotlib 和许多其他有用的工具。

Linux：使用你的包管理器

如果你是用 Linux，你可能更倾向于使用包管理器。matplotlib 是用于多数主流 Linux 发行版的包。

- Debian / Ubuntu： `sudo apt-get install python-matplotlib`
- Fedora / Redhat： `sudo yum install python-matplotlib`

Mac OSX：使用 `pip`

如果你使用 MacOS，你可以使用 Python 标准安装程序 `pip` 来安装 matplotlib 二进制。参见安装 MacOS 二进制轮子。

Windows

如果你还没有安装 Python，我们建议使用兼容 SciPy 技术栈的 Python 分发版本，如 WinPython，Python(x, y)，Enthought Canopy 或 Continuum Anaconda，它们含有 matplotlib 和它的许多依赖，并预装了其他有用的软件包。

对于 Python 的标准安装，可以使用 `pip` 安装 matplotlib ：

```
python -m pip install -U pip setuptools
python -m pip install matplotlib
```

如果没有为所有用户安装 Python 2.7 或 3.4，则需要安装 Microsoft Visual C++ 2008（对于 Python 2.7 为 64 位或 32 位）或 Microsoft Visual C++ 2010（对于 Python 3.4 为 64 位或 32 位）再分发包。

Matplotlib 依赖于 Pillow 来读取和保存 JPEG，BMP 和 TIFF 图像文件。 Matplotlib 需要 MiKTeX 和 GhostScript 来使用LaTeX渲染文本。动画模块需要 FFmpeg，avconv，mencoder 或 ImageMagick。

以下后端应该开箱即用：agg，tkagg，ps，pdf 和 svg。对于其他后端，你可能需要安装 pycairo，PyQt4，PyQt5，PySide，wxPython，PyGTK，Tornado 或 GhostScript。

TkAgg 可能是来自标准 Python shell 或 IPython 的，用于交互式的最佳后端。它被启用为官方二进制文件的默认后端。 Windows 不支持 GTK3。

PyPI 下载页面上的 Windows 轮子（ `* .whl` ）不包含测试数据或示例代码。如果你想尝试 matplotlib 源代码中的许多演示，请下载 `*.tar.gz` 文件并查看 `examples` 子目录。要运行测试套件，请将源代码发行版中的 `lib\matplotlib\tests` 和 `lib\mpl_toolkits\tests` 目录分别复制到 `sys.prefix\Lib\site-packages\matplotlib` 和 `sys.prefix\Lib\site-pac`，并安装 nose，mock，Pillow，MiKTeX，GhostScript，ffmpeg，avconv，mencoder，ImageMagick 和 Inkscape。

从源码安装

如果你有兴趣为 matplotlib 开发做贡献，运行最新的源代码，或者只是想自己构建一切，从源代码构建 matplotlib 并不困难。 从 PyPI 文件页面抓取最新的 `tar.gz` 发布文件，或者如果你想开发 matplotlib 或只需要最新的 bug 修复版本，获取最新的 git 版本，请见从 git 安装。

源代码遵守标准环境变量 CC，CXX，PKG_CONFIG。 这意味着如果你的工具链有前缀，你可以设置它们。 这可以用于交叉编译。

```
export CC=x86_64-pc-linux-gnu-gcc export CXX=x86_64-pc-linux-gnu
-g++ export PKG_CONFIG=x86_64-pc-linux-gnu-pkg-config
```

一旦你满足的了面的具体需求（主要是 Python、NumPy、libpng 和 freetype），你就可以构建 matplotlib 了：

```
cd matplotlib
python setup.py build
python setup.py install
```

我们提供与 `setup.py` 一起使用的 `setup.cfg` 文件，你可以使用它来自定义构建过程。 例如，要使用的默认后端，是否安装 matplotlib 附带的某些可选库，等等。这个文件会对那些包装 matplotlib 的东西特别有用。

如果已经为非标准设施安装了必备组件，并需要通知 matplotlib 它们在哪里，请编辑 `setupext.py` 并将基本路径添加为 `sys.platform` 的 `basedir` 字典条目。例如，如果某些所需库的头文件位于 `/some/path/include/someheader.h` 中，请在你的平台的 `basedir` 列表中输入 `/some/path` 。

构建需求

这些是外部软件包，你需要在安装 matplotlib 之前安装它们。 如果你在 OSX 上构建，请参阅在 OSX 上构建。 如果你在 Windows 上构建，请参阅在 Windows 上构建。 如果在 Linux 上使用软件包管理器安装依赖项，则除了库本身之外，还可能需要安装开发包（查找 `-dev` 后缀）。

所需依赖

Python 2.7，3.4，3.5 或 3.6

下载 Python

NumPy 1.7.1（或更新）

Python 的数组支持（下载 NumPy）

setuptools

setuptools 为 Python 包安装提供扩展

dateutil 1.1 或更新

为 Python 时间日期的处理提供扩展。如果使用了 `pip` ， `easy_install` 或者从源码安装，安装器会尝试从 PyPI 下载并安装 `python_dateutil` 。

pyparsing

需要为 matplotlib 的 mathtext 数学渲染提供支持。如果使用了 `pip` ， `easy_install` 或者从源码安装，安装器会尝试从 PyPI 下载并安装 `pyparsing` 。

libpng 1.2 （或更新）

用于加载和保存 PNG 文件（下载）。libong 需要 zlib。

pytz

用于操作时区感知的日期时间。https://pypi.python.org/pypi/pytz

FreeType 2.3 或更新

用于读取 TrueType 字体文件。如果使用了 `pip` ， `easy_install` 或者从源码安装，安装器会尝试从预期位置定位 FreeType。如果找病毒奥，尝试安装 pkg-config，用于寻找所需非 Python 库的工具。

cycler 0.10.1 或更新

可组合的循环类，用于构造 style-cycle。

six

需要用于 Python 2 和 3 之间的兼容性。

Python 2 的依赖

functools32

需要用于 Python 2.7 上的兼容性。

subprocess32

可选，仅用于 Unix。 `subprocess` 标准库从 3.2+ 到 2.7 的 Backport。它提供了更好的错误信息和超时支持。

可选的 GUI 框架

这些是可选软件包，你可能希望安装这些软件包来使 用matplotlib 和用户界面工具包。 有关 matplotlib 可选后端和所提供功能的更多详细信息，请参阅什么是后端。

tk 8.3 或更新，不包括 8.6.0 和 8.6.1

TkAgg 后端使用的 TCL/Tk 窗口控件库。

版本 8.6. 0和 8.6.1 已知有问题，当以错误的顺序关闭多个窗口时可能导致段错误。

pyqt 4.4 或更新

Qt4 控件库的 Python 包装，用于 Qt4Agg 后端。

pygtk 2.4 或更新

GTK 控件库的 Python 包装，用于 GTK 或者 GTKAgg 后端。

wxpython 2.8 或更新

wx 控件库的 Python 包装，用于 WX 或 WXAgg 后端。

可选的外部程序

ffmpeg/avconv 或 mencoder

需要用于动画模块，将输出保存为电影格式。

ImageMagick

需要用于动画模块，能够保存 GIF 动画。

可选依赖

Pillow

如果安装了 Pillow，matplotlib 可以读取或写入大部分图像文件格式。

pkg-config

用于寻找所需非 Python 库的工具。并不是严格需要它，但是如果库和头文件不在预期位置，可以使安装更加便捷。

matplotlib 自带的所需库

agg 2.4

C++ 渲染引擎。 matplotlib 静态链接到 agg 模板源码，所以它除了 matplotlib 之外，不会影响你的系统的任何东西。

qhull 2012.1

用于计算 Delaunay 三角测量的库。

ttconv

TureType 字体工具。

在 Linux 上构建

使用你的系统包管理器来安装依赖最为简单。

如果你使用 Debian/Ubuntu，可以使用以下命令在获取需要用于构建 matplotlib 的所有依赖：

```
sudo apt-get build-dep python-matplotlib
```

如果你使用 Fedora/RedHat，你可以使用以下命令：

```
su -c "yum-builddep python-matplotlib"
```

这不会构建 matplotlib，但这会安装所需依赖。这会使从源码构建变得容易。

在 OSX 上构建

由于可以获取 `libpng` 和 `freetype` 需求（darwinports，fink，/usr/X11R6）的不同位置，不同的架构（例如 x86，ppc，universal）和不同的 OSX 版本 10.4 和 10.5），OSX 的构建情况很复杂。我们建议你使用我们对 OSX 版本所做的方式来

构建：从 `tarball` 或 `git` 仓库获取源代码，并按照 `README.osx` 中的说明进行操作。

在 **Windows** 上构建

https://www.python.org 上发布的 Python ，使用 VS2008 编译 3.3 之前的版本，使用 VS2010 编译 3.3，并且使用 VS2015 编译 3.5 和 3.6。 建议使用相同的编译器编译 Python 扩展。

由于没有规范的 Windows 包管理器，从源代码构建 `freetype` ， `zlib` 和 `libpng` 的方法被记录为 `matplotlib-winbuild` 中的构建脚本。

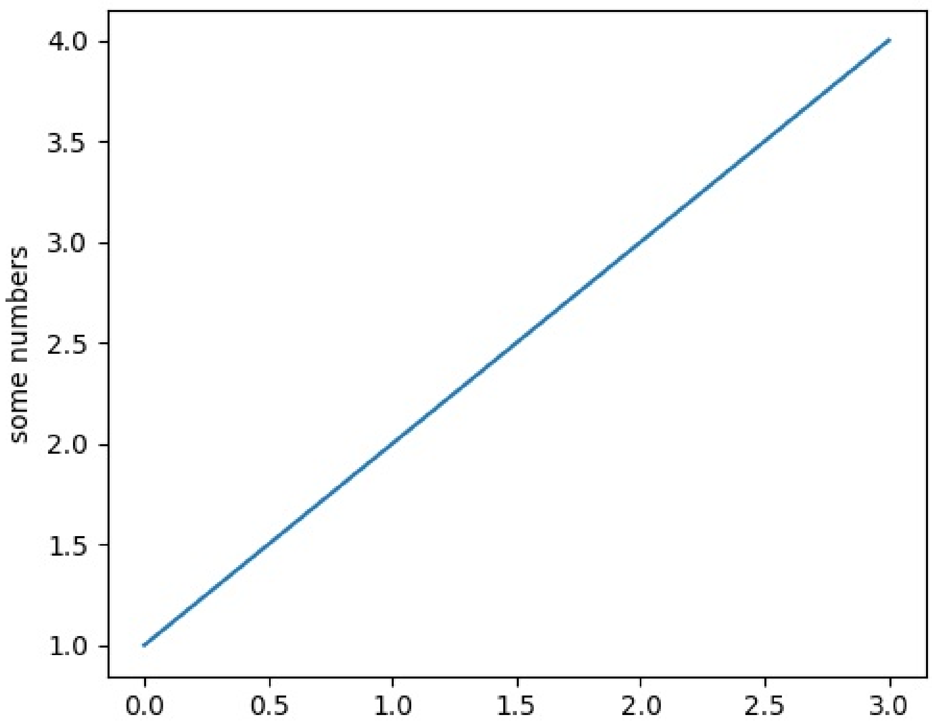
教程

pyplot 教程

> 原文：Pyplot tutorial
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

`matplotlib.pyplot` 是一个命令风格函数的集合，使 `matplotlib` 的机制更像 MATLAB。 每个绘图函数对图形进行一些更改：例如，创建图形，在图形中创建绘图区域，在绘图区域绘制一些线条，使用标签装饰绘图等。
在 `matplotlib.pyplot` 中，各种状态跨函数调用保存，以便跟踪诸如当前图形和绘图区域之类的东西，并且绘图函数始终指向当前轴域（请注意，这里和文档中的大多数位置中的『轴域』（axes）是指图形的一部分（两条坐标轴围成的区域），而不是指代多于一个轴的严格数学术语）。

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4])
plt.ylabel('some numbers')
plt.show()
```



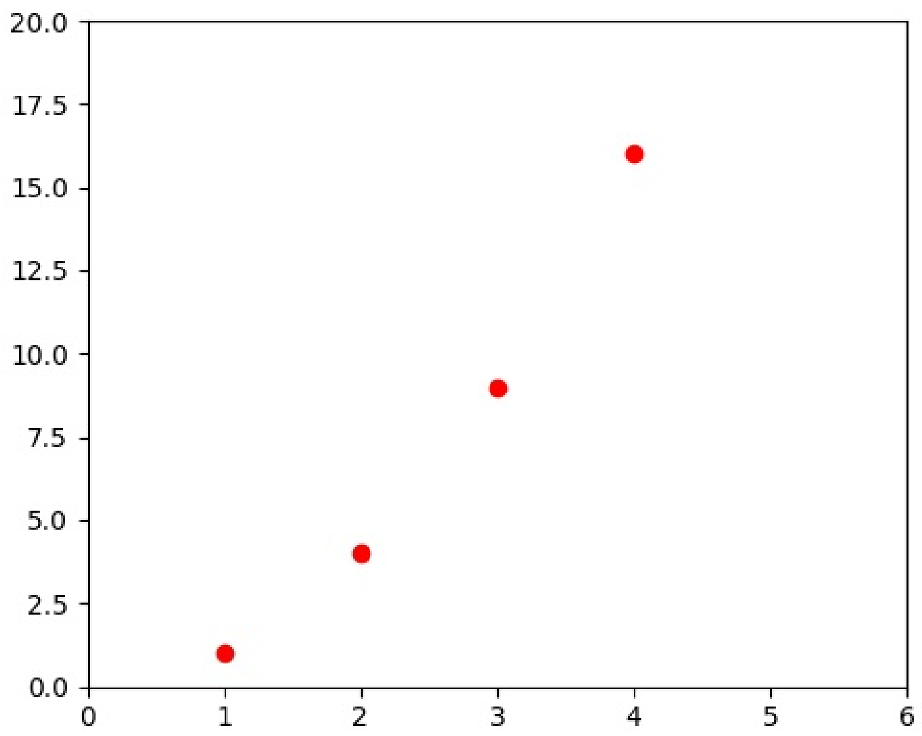
你可能想知道为什么 `x` 轴的范围为 `0-3` ，`y` 轴的范围为 `1-4` 。如果你向 `plot()` 命令提供单个列表或数组，则 `matplotlib` 假定它是一个 `y` 值序列，并自动为你生成 `x` 值。由于 python 范围从 0 开始，默认 `x` 向量具有与 `y` 相同的长度，但从 0 开始。因此 `x` 数据是 `[0,1,2,3]` 。

`plot()` 是一个通用命令，并且可接受任意数量的参数。例如，要绘制 `x` 和 `y` ，你可以执行命令：

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
```

对于每个 `x,y` 参数对，有一个可选的第三个参数，它是指示图形颜色和线条类型的格式字符串。格式字符串的字母和符号来自 MATLAB，并且将颜色字符串与线型字符串连接在一起。默认格式字符串为 `"b-"` ，它是一条蓝色实线。例如，要绘制上面的红色圆圈，你需要执行：

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4], [1,4,9,16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```



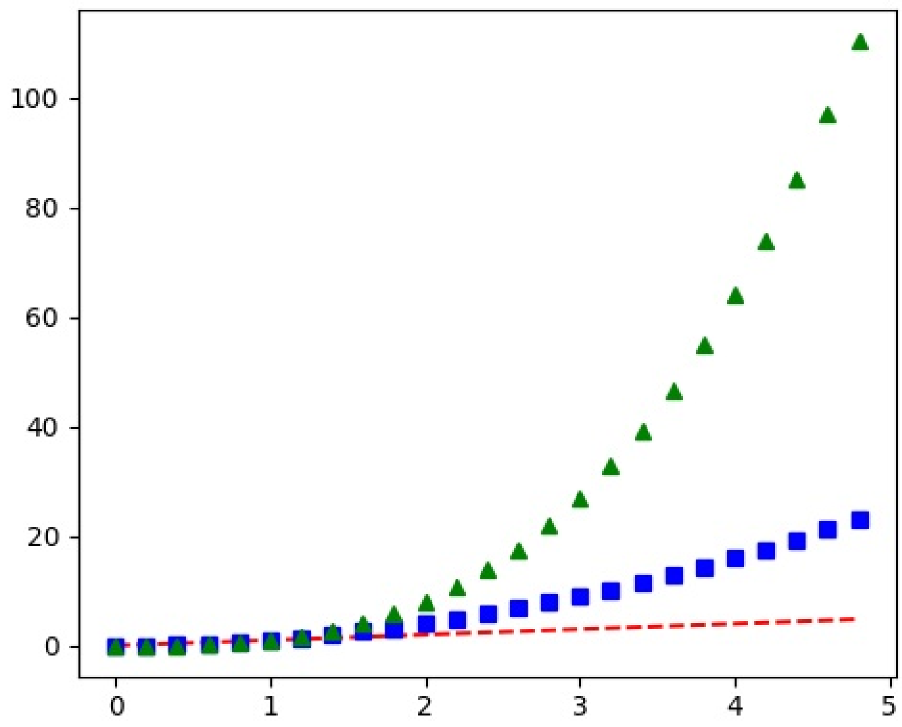
有关线型和格式字符串的完整列表，请参见 `plot()` 文档。上例中的 `axis()` 命令接收 `[xmin，xmax，ymin，ymax]` 的列表，并指定轴域的可视区域。

如果 `matplotlib` 仅限于使用列表，它对于数字处理是相当无用的。 一般来说，你可以使用 `numpy` 数组。 事实上，所有序列都在内部转换为 `numpy` 数组。 下面的示例展示了使用数组和不同格式字符串，在一条命令中绘制多个线条。

```
import numpy as np
import matplotlib.pyplot as plt

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```



控制线条属性

线条有许多你可以设置的属性： `linewidth` ， `dash style` ， `antialiased` 等，请参见 `matplotlib.lines.Line2D` 。 有几种方法可以设置线属性：

- 使用关键字参数：

  ```
  plt.plot(x, y, linewidth=2.0)
  ```

- 使用 `Line2D` 实例的 `setter` 方法。 `plot` 返回 `Line2D` 对象的列表，例如 `line1,line2 = plot(x1，y1，x2，y2)` 。在下面的代码中，我们假设只有一行，返回的列表长度为 1。我们对 `line` 使用元组解构，得到该列表的第一个元素：

```
line, = plt.plot(x, y, '-')
line.set_antialiased(False) # turn off antialising
```

- 使用 `setp()` 命令。 下面的示例使用 MATLAB 风格的命令来设置线条列表上的多个属性。 `setp` 使用对象列表或单个对象透明地工作。 你可以使用 python 关键字参数或 MATLAB 风格的字符串/值对：

```
lines = plt.plot(x1, y1, x2, y2)
# 使用关键字参数
plt.setp(lines, color='r', linewidth=2.0)
# 或者 MATLAB 风格的字符串值对
plt.setp(lines, 'color', 'r', 'linewidth', 2.0)
```

下面是可用的 `Line2D` 属性。

属性	值类型
`alpha`	浮点值
animated	[True / False]
`antialiased or aa`	`[True / False]`
clip_box	matplotlib.transform.Bbox 实例
`clip_on`	`[True / False]`
clip_path	Path 实例， Transform ，以及 Patch 实例
`color or c`	任何 `matplotlib` 颜色
contains	命中测试函数
`dash_capstyle`	`['butt' / 'round' / 'projecting']`
dash_joinstyle	['miter' / 'round' / 'bevel']
`dashes`	以点为单位的连接/断开墨水序列
data	(np.array xdata, np.array ydata)
`figure`	`matplotlib.figure.Figure` 实例
label	任何字符串
`linestyle` or `ls`	['-' / '--' / '-.' / ':' / 'steps' / .

linewidth or lw	以点为单位的浮点值
lod	[True / False]
marker	['+' / ',' / '.' / '1' / '2' / '3' / '
markeredgecolor or mec	任何 matplotlib 颜色
markeredgewidth or mew	以点为单位的浮点值
markerfacecolor or mfc	任何 matplotlib 颜色
markersize or ms	浮点值
markevery	[None / 整数值 / (startind, stride)]
picker	用于交互式线条选择
pickradius	线条的拾取选择半径
solid_capstyle	['butt' / 'round' / 'projecting']
solid_joinstyle	['miter' / 'round' / 'bevel']
transform	matplotlib.transforms.Transform 实例
visible	[True / False]
xdata	np.array
ydata	np.array
zorder	任何数值

要获取可设置的线条属性的列表，请以一个或多个线条作为参数调用 step() 函数

```
In [69]: lines = plt.plot([1, 2, 3])

In [70]: plt.setp(lines)
  alpha: float
  animated: [True | False]
  antialiased or aa: [True | False]
  ...snip
```

处理多个图形和轴域

MATLAB 和 pyplot 具有当前图形和当前轴域的概念。 所有绘图命令适用于当前轴域。 函数 gca() 返回当前轴域（一个 matplotlib.axes.Axes 实例），gcf() 返回当前图形（ matplotlib.figure.Figure 实例）。 通常，你不必担心这一点，因为它都是在幕后处理。 下面是一个创建两个子图的脚本。

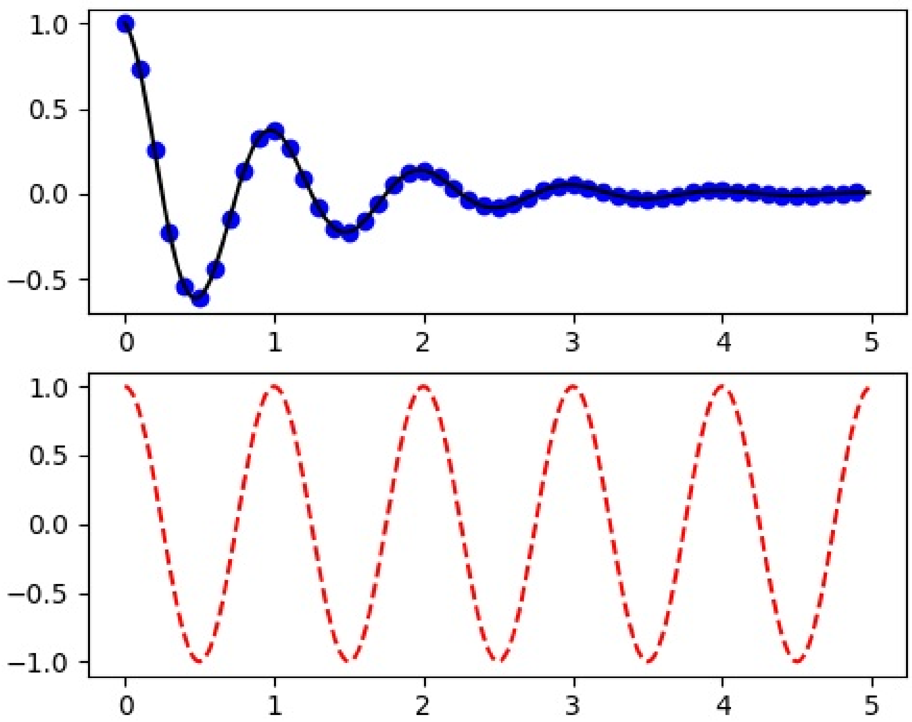
```
import numpy as np
import matplotlib.pyplot as plt

def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure(1)
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```



这里的 `figure()` 命令是可选的，因为默认情况下将创建 `figure(1)` ，如果不手动指定任何轴域，则默认创建 `subplot(111)` 。 `subplot()` 命令指定 `numrows` ， `numcols` ， `fignum` ，其中 `fignum` 的范围是从 `1` 到 `numrows * numcols` 。 如果 `numrows * numcols <10` ，则 `subplot` 命令中的逗号是可选的。 因此，子图 `subplot(211)` 与 `subplot(2, 1, 1)` 相同。 你可以创建任意数量的子图和轴域。 如果要手动放置轴域，即不在矩形网格上，请使用 `axes()` 命令，该命令允许你将 `axes([left, bottom, width, height])` 指定为位置，其中所有值都使

用小数（0 到 1）坐标。 手动放置轴域的示例请参见 `pylab_examples` 示例代码： `axes_demo.py` ，具有大量子图的示例请参见 `pylab_examples` 示例代码： `subplots_demo.py` 。

你可以通过使用递增图形编号多次调用 `figure()` 来创建多个图形。 当然，每个数字可以包含所需的轴和子图数量：

```
import matplotlib.pyplot as plt
plt.figure(1)                # 第一个图形
plt.subplot(211)             # 第一个图形的第一个子图
plt.plot([1, 2, 3])
plt.subplot(212)             # 第一个图形的第二个子图
plt.plot([4, 5, 6])


plt.figure(2)                # 第二个图形
plt.plot([4, 5, 6])          # 默认创建 subplot(111)

plt.figure(1)                # 当前是图形 1，subplot(212)
plt.subplot(211)             # 将第一个图形的 subplot(211) 设为当前
子图
plt.title('Easy as 1, 2, 3') # 子图 211 的标题
```

你可以使用 `clf()` 清除当前图形，使用 `cla()` 清除当前轴域。 如果你搞不清在幕后维护的状态（特别是当前的图形和轴域），不要绝望：这只是一个面向对象的 API 的简单的状态包装器，你可以使用面向对象 API（见艺术家教程）。

如果你正在制作大量的图形，你需要注意一件事：在一个图形用 `close()` 显式关闭之前，该图所需的内存不会完全释放。 删除对图形的所有引用，和/或使用窗口管理器杀死屏幕上出现的图形的窗口是不够的，因为在调用 `close()` 之前， `pyplot` 会维护内部引用。

处理文本

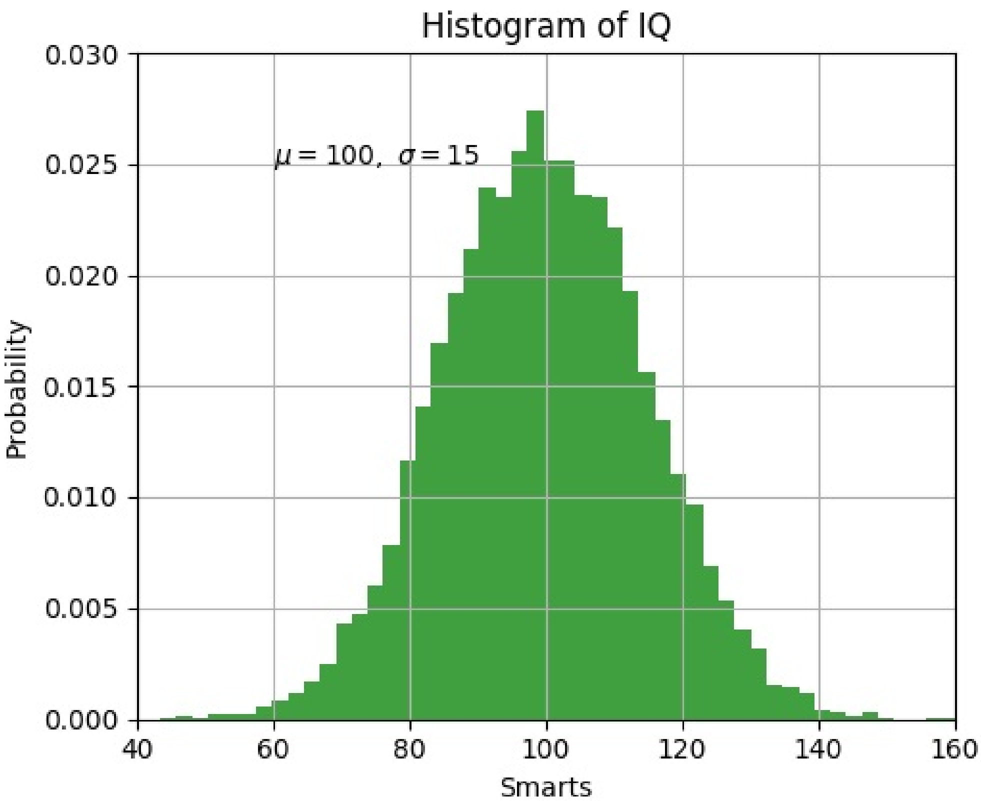
`text()` 命令可用于在任意位置添加文本， `xlabel()` ， `ylabel()` 和 `title()` 用于在指定的位置添加文本（详细示例请参阅文本介绍）。

```
import numpy as np
import matplotlib.pyplot as plt

mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

# 数据的直方图
n, bins, patches = plt.hist(x, 50, normed=1, facecolor='g', alph
a=0.75)


plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```



所有的 `text()` 命令返回一个 `matplotlib.text.Text` 实例。与上面一样，你可以通过将关键字参数传递到 `text` 函数或使用 `setp()` 来自定义属性：

```
t = plt.xlabel('my data', fontsize=14, color='red')
```

这些属性的更详细介绍请见文本属性和布局。

在文本中使用数学表达式

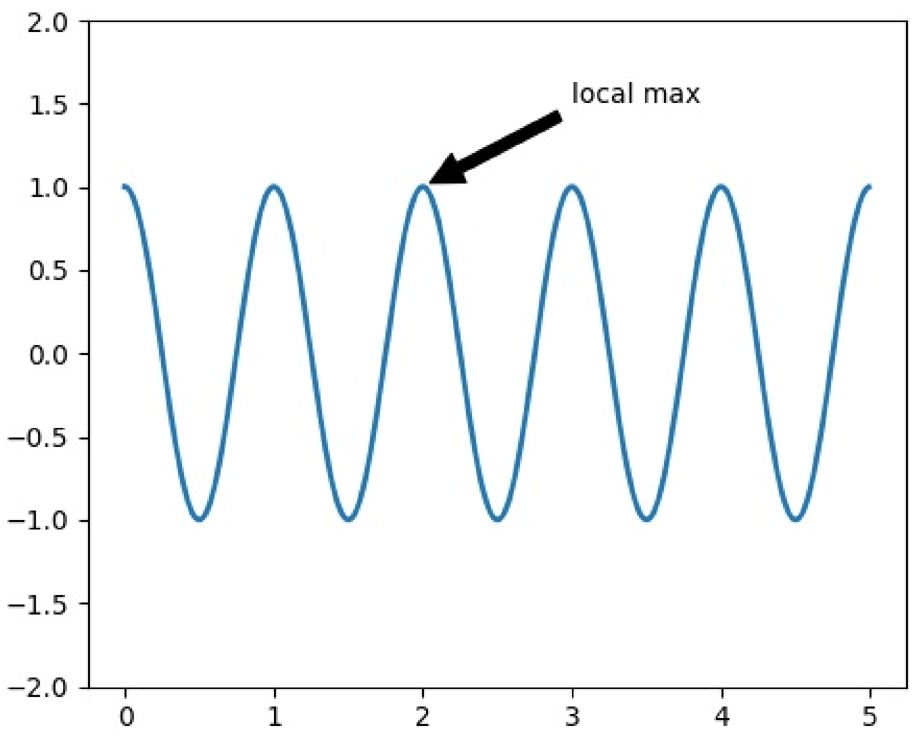
`matplotlib` 在任何文本表达式中接受 TeX 方程表达式。 例如，要在标题中写入表达式，可以编写一个由美元符号包围的 TeX 表达式：

```
plt.title(r'$\sigma_i=15$')
```

标题字符串之前的 `r` 很重要 - 它表示该字符串是一个原始字符串，而不是将反斜杠作为 python 转义处理。 `matplotlib` 有一个内置的 TeX 表达式解析器和布局引擎，并且自带了自己的数学字体 - 详细信息请参阅编写数学表达式。 因此，你可以跨平台使用数学文本，而无需安装 TeX。 对于安装了 LaTeX 和 `dvipng` 的用户，还可以使用 LaTeX 格式化文本，并将输出直接合并到显示图形或保存的 postscript 中 - 请参阅使用 LaTeX 进行文本渲染。

标注文本

上面的 `text()` 基本命令将文本放置在轴域的任意位置。 文本的一个常见用法是对图的某些特征执行标注，而 `annotate()` 方法提供一些辅助功能，使标注变得容易。 在标注中，有两个要考虑的点：由参数 `xy` 表示的标注位置和 `xytext` 表示的文本位置。 这两个参数都是 `(x, y)` 元组。



在此基本示例中，`xy`（箭头提示）和`xytext`（文本）都位于数据坐标中。 有多种其他坐标系可供选择 - 详细信息请参阅标注文本和标注轴域。 更多示例可以在`pylab_examples`示例代码：`annotation_demo.py`中找到。

对数和其它非线性轴

`matplotlib.pyplot`不仅支持线性轴刻度，还支持对数和对数刻度。 如果数据跨越许多数量级，通常会使用它。 更改轴的刻度很容易：

```
plt.xscale('log')
```

下面示例显示了四个图，具有相同数据和不同刻度的`y`轴。

```
import numpy as np
import matplotlib.pyplot as plt

# 生成一些区间 [0, 1] 内的数据
y = np.random.normal(loc=0.5, scale=0.4, size=1000)
y = y[(y > 0) & (y < 1)]
y.sort()
x = np.arange(len(y))

# 带有多个轴域刻度的 plot
plt.figure(1)

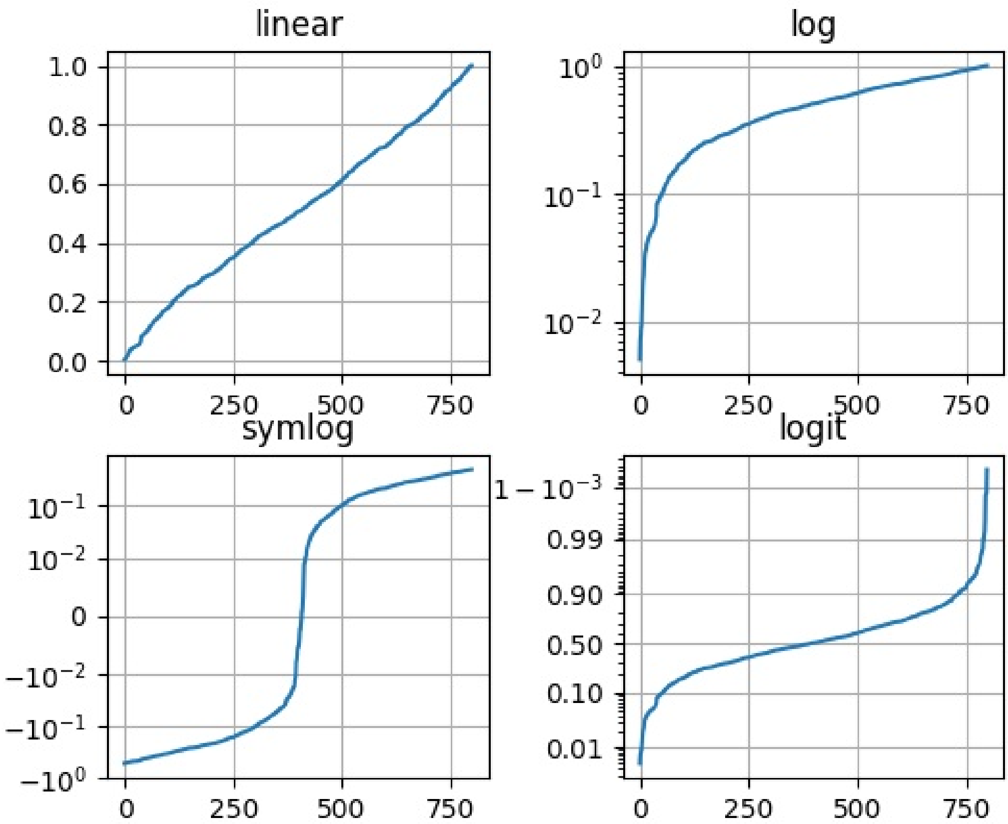
# 线性
plt.subplot(221)
plt.plot(x, y)
plt.yscale('linear')
plt.title('linear')
plt.grid(True)


# 对数
plt.subplot(222)
plt.plot(x, y)
plt.yscale('log')
plt.title('log')
plt.grid(True)


# 对称的对数
plt.subplot(223)
plt.plot(x, y - y.mean())
plt.yscale('symlog', linthreshy=0.05)
plt.title('symlog')
plt.grid(True)

# logit
plt.subplot(224)
plt.plot(x, y)
plt.yscale('logit')
plt.title('logit')
plt.grid(True)

plt.show()
```



还可以添加自己的刻度，详细信息请参阅向 `matplotlib` 添加新的刻度和投影。

图像教程

> 原文：Image tutorial
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

启动命令

首先，让我们启动 IPython。 它是 Python 标准提示符的最好的改进，它与 Matplotlib 配合得相当不错。 在 shell 或 IPython Notebook 上都可以启动 IPython。

随着 IPython 启动，我们现在需要连接到 GUI 事件循环。 它告诉 IPython 在哪里（以及如何显示）绘图。 要连接到 GUI 循环，请在 IPython 提示符处执行 `%matplotlib` 魔法。 在 IPython 的 GUI 事件循环文档中有更多的细节。

如果使用 IPython Notebook，可以使用相同的命令，但人们通常以特定参数使用 `%matplotlib` ：

```
In [1]: %matplotlib inline
```

这将打开内联绘图，绘图图形将显示在笔记本中。 这对交互性有很重要的影响。 对于内联绘图，在单元格下方的单元格中输出绘图的命令不会影响绘图。 例如，从创建绘图的单元格下面的单元格更改颜色表是不可能的。 但是，对于其他后端，例如 qt4，它们会打开一个单独的窗口，那些创建绘图的单元格下方的单元格将改变绘图 - 它是一个内存中的活对象。

本教程将使用 `matplotlib` 的命令式绘图接口 `pyplot` 。 该接口维护全局状态，并且可用于简单快速地尝试各种绘图设置。 另一种是面向对象的接口，这也非常强大，一般更适合大型应用程序的开发。 如果你想了解面向对象接口，使用上的常见问题是一个用于起步的不错的页面。 现在，让我们继续使用命令式方式：

```
In [2]: import matplotlib.pyplot as plt
In [3]: import matplotlib.image as mpimg
In [4]: import numpy as np
```

将图像数据导入到 NumPy 数组

加载图像数据由 Pillow 库提供支持。 本来， `matplotlib` 只支持 PNG 图像。 如果本机读取失败，下面显示的命令会回退到 Pillow。

此示例中使用的图像是 PNG 文件，但是请记住你自己的数据的 Pillow 要求。

下面是我们要摆弄的图片：



它是一个 24 位 RGB PNG 图像（每个 R，G，B 为 8 位）。根据你获取数据的位置，你最有可能遇到的其他类型的图像是 RGBA 图像，拥有透明度或单通道灰度（亮度）的图像。你可以右键单击它，选择 `Save image as`（另存为）为本教程的剩余部分下载到你的计算机。

现在我们开始...

```
In [5]: img=mpimg.imread('stinkbug.png')
Out[5]:
array([[[ 0.40784314,  0.40784314,  0.40784314],
        [ 0.40784314,  0.40784314,  0.40784314],
        [ 0.40784314,  0.40784314,  0.40784314],
        ...,
        [ 0.42745098,  0.42745098,  0.42745098],
        [ 0.42745098,  0.42745098,  0.42745098],
        [ 0.42745098,  0.42745098,  0.42745098]],

       ...,
       [[ 0.44313726,  0.44313726,  0.44313726],
        [ 0.4509804 ,  0.4509804 ,  0.4509804 ],
        [ 0.4509804 ,  0.4509804 ,  0.4509804 ],
        ...,
        [ 0.44705883,  0.44705883,  0.44705883],
        [ 0.44705883,  0.44705883,  0.44705883],
        [ 0.44313726,  0.44313726,  0.44313726]]], dtype=float32
)
```

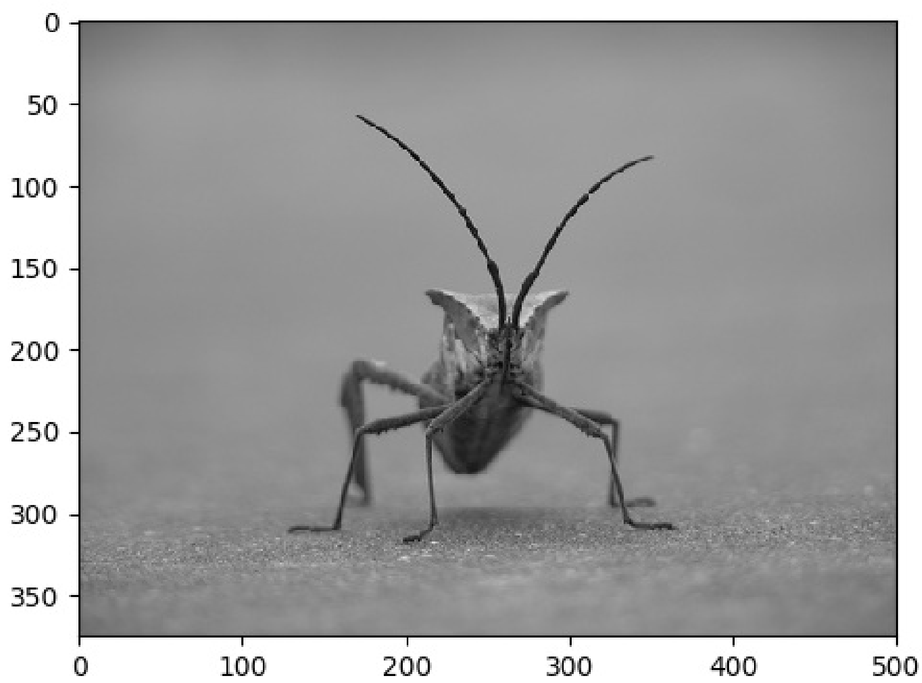
注意这里的 `dtype` - `float32` 。 Matplotlib 已将每个通道的8位数据重新定标为 0.0 和 1.0 之间的浮点数。 作为旁注，Pillow 可以使用的唯一数据类型是 `uint8` 。 Matplotlib 绘图可以处理 `float32` 和 `uint8` ，但是对于除 PNG 之外的任何格式的图像，读取/写入仅限于 `uint8` 数据。 为什么是 8 位呢？ 大多数显示器只能渲染每通道 8 位的颜色渐变。 为什么他们只能渲染每通道 8 位呢？ 因为这会使所有人的眼睛可以看到。 更多信息请见（从摄影的角度）：Luminous Landscape 位深度教程。

每个内部列表表示一个像素。 这里，对于 RGB 图像，有 3 个值。 由于它是一个黑白图像，R，G 和 B 都是类似的。 RGBA（其中 A 是阿尔法或透明度）对于每个内部列表具有 4 个值，而且简单亮度图像仅具有一个值（因此仅是二维数组，而不是三维数组）。 对于 RGB 和 RGBA 图像， `matplotlib` 支持 `float32` 和 `uint8` 数据类型。 对于灰度， `matplotlib` 只支持 `float32` 。 如果你的数组数据不符合这些描述之一，则需要重新缩放它。

将 NumPy 数组绘制为图像

所以，你将数据保存在一个 `numpy` 数组（通过导入它，或生成它）。 让我们渲染它吧。 在 Matplotlib 中，这是使用 `imshow()` 函数执行的。 这里我们将抓取 `plot` 对象。 这个对象提供了一个简单的方法来从提示符处理绘图。

```
In [6]: imgplot = plt.imshow(img)
```



你也可以绘制任何 NumPy 数组。

对图像绘图应用伪彩色方案

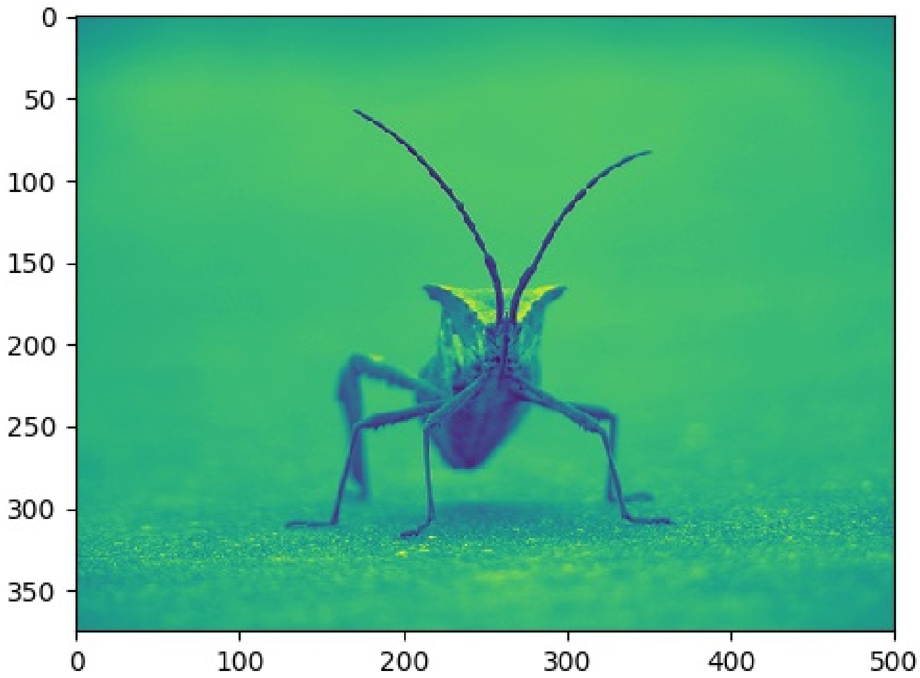
伪彩色可以是一个有用的工具，用于增强对比度和更易于可视化你的数据。 这在使用投影仪对你的数据进行演示时尤其有用 - 它们的对比度通常很差。

伪彩色仅与单通道，灰度，亮度图像相关。 我们目前有一个RGB图像。 由于R，G和 B 都是相似的（见上面或你的数据），我们可以只选择一个通道的数据：

```
In [7]: lum_img = img[:,:,0]
```

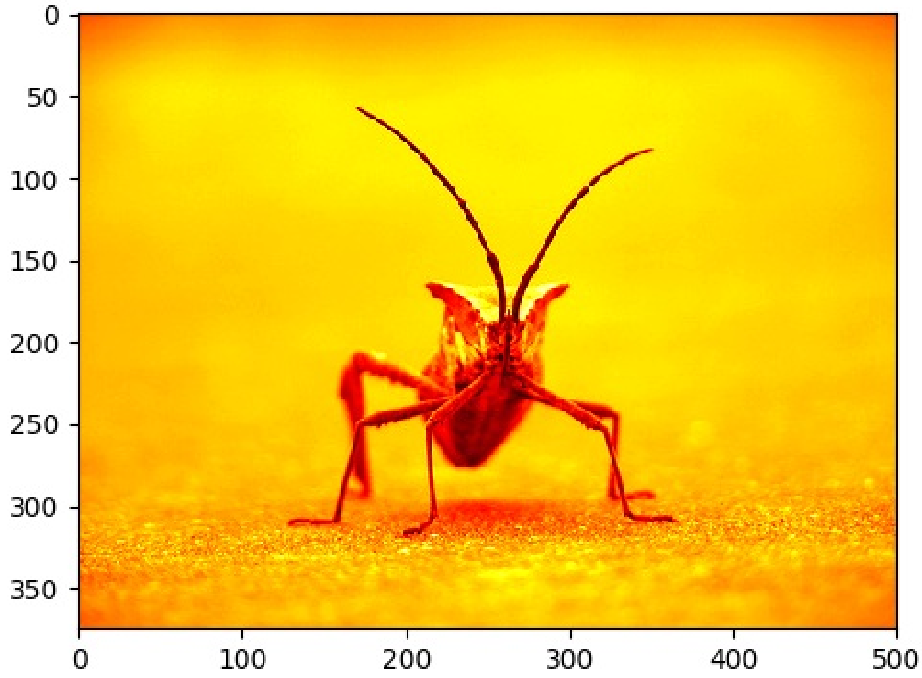
这是数组切片，更多信息请见NumPy 教程。

```
In [8]: plt.imshow(lum_img)
```



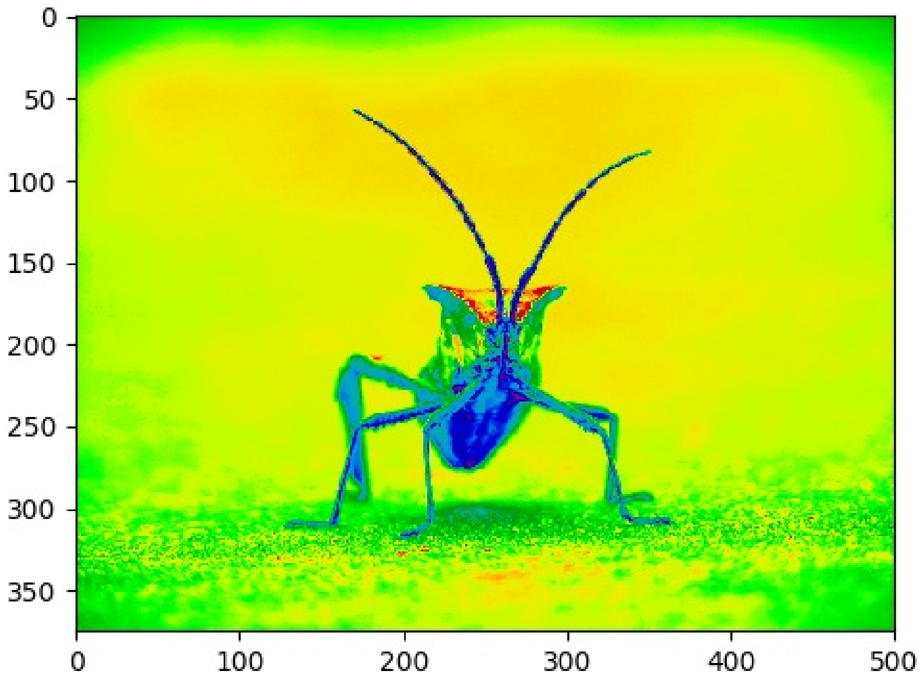
现在，亮度（2D，无颜色）图像应用了默认颜色表（也称为查找表，LUT）。 默认值称为 `jet` 。 有很多其他方案可以选择。

```
In [9]: plt.imshow(lum_img, cmap="hot")
```



请注意，你还可以使用`set_cmap()`方法更改现有绘图对象上的颜色：

```
In [10]: imgplot = plt.imshow(lum_img)
In [11]: imgplot.set_cmap('spectral')
```



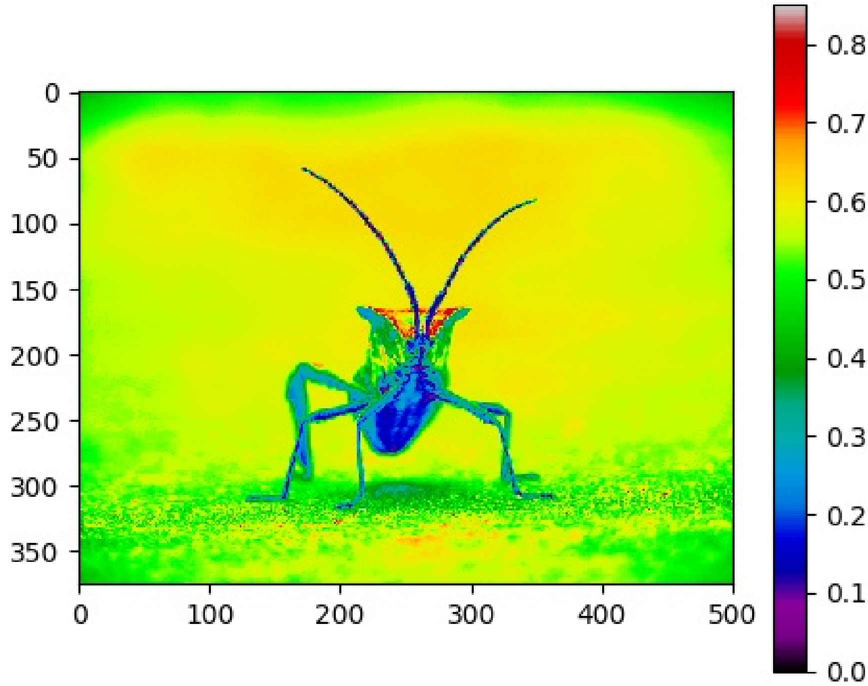
> 注
>
> 但是，请记住，在带有内联后端的 IPython notebook 中，你不能对已经渲染的绘图进行更改。 如果你在一个单元格中创建了 `imgplot` ，你不能在以后的单元格中调用 `set_cmap()` ，并且改变前面的绘图。 请确保你在相同单元格中一起输入这些命令。 `plt` 命令不会更改先前单元格的绘图。

有许多可选的其它颜色表，请见颜色表的列表和图像。

颜色刻度参考

了解颜色代表什么值对我们很有帮助。 我们可以通过添加颜色条来做到这一点。

```
In [12]: imgplot = plt.imshow(lum_img)
In [13]: plt.colorbar()
```

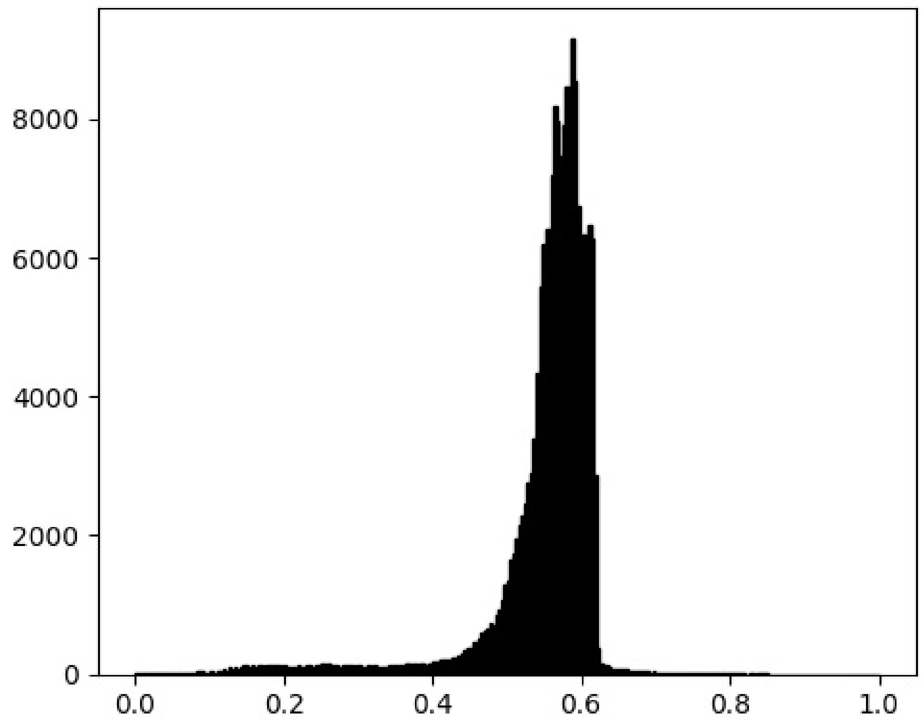


这会为你现有的图形添加一个颜色条。 如果你更改并切换到不同的颜色映射，则不会自动更改 - 你必须重新创建绘图，并再次添加颜色条。

检查特定数据范围

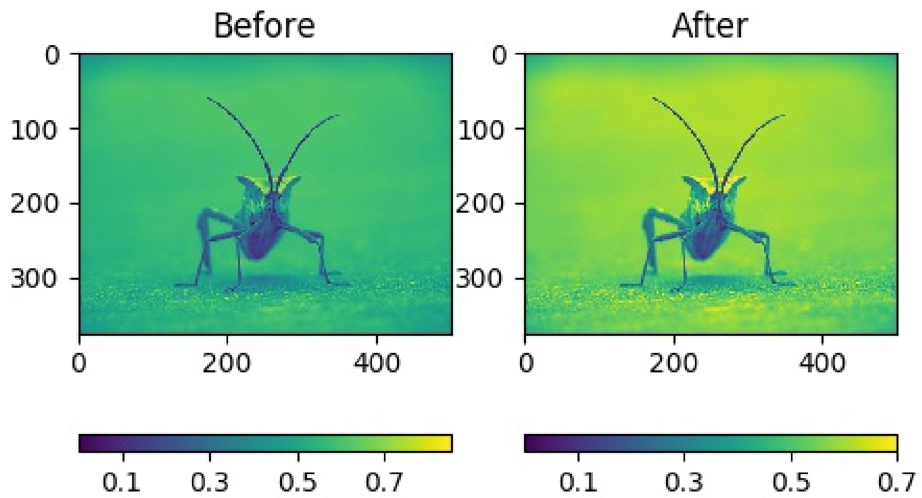
有时，你想要增强图像的对比度，或者扩大特定区域的对比度，同时牺牲变化不大，或者无所谓的颜色细节。 找到有趣区域的最好工具是直方图。 要创建我们的图像数据的直方图，我们使用 `hist()` 函数。

```
In [14]: plt.hist(lum_img.ravel(), bins=256, range=(0.0, 1.0), f
c='k', ec='k')
```



通常，图像的『有趣』部分在峰值附近，你可以通过剪切峰值上方和/或下方的区域获得额外的对比度。 在我们的直方图中，看起来最大值处没有太多有用的信息（图像中有很多不是白色的东西）。 让我们调整上限，以便我们有效地『放大』直方图的一部分。 我们通过将 `clim` 参数传递给 `imshow` 来实现。 你也可以通过对图像绘图对象调用 `set_clim()` 方法来做到这一点，但要确保你在使用 IPython Notebook 的时候，和 `plot` 命令在相同的单元格中执行 - 它不会改变之前单元格的图。

```
In [15]: imgplot = plt.imshow(lum_img, clim=(0.0, 0.7))
```

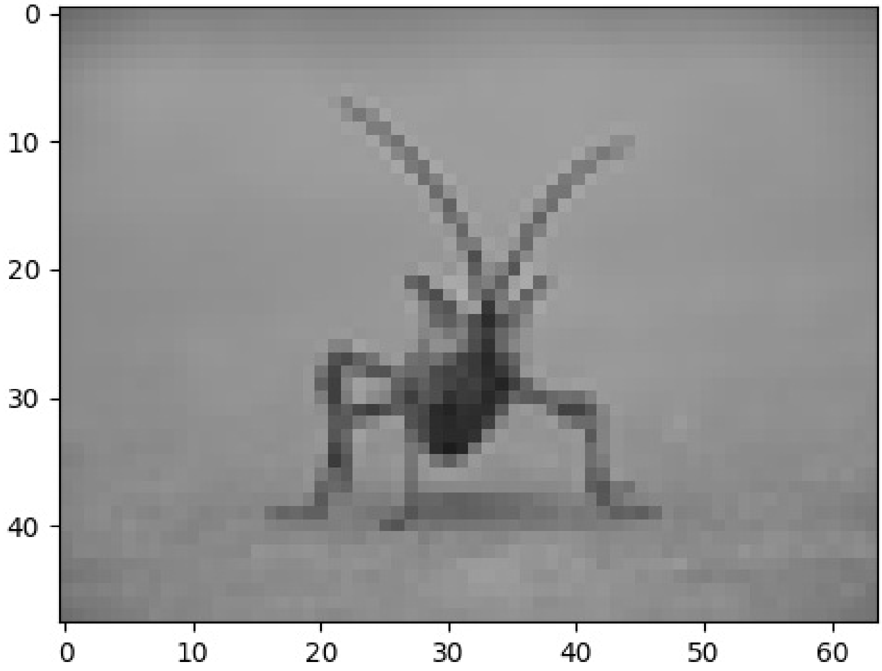


数组插值方案

插值根据不同的数学方案计算像素『应有』的颜色或值。 发生这种情况的一个常见的场景是调整图像的大小。 像素的数量会发生变化，但你想要相同的信息。 由于像素是离散的，因此存在缺失的空间。 插值就是填补这个空间的方式。 这就是当你放大图像时，你的图像有时会出来看起来像素化的原因。 当原始图像和扩展图像之间的差异较大时，效果更加明显。 让我们加载我们的图像并缩小它。 我们实际上正在丢弃像素，只保留少数几个像素。 现在，当我们绘制它时，数据被放大为你屏幕的大小。 由于旧的像素不再存在，计算机必须绘制像素来填充那个空间。

我们将使用用来加载图像的 Pillow 库来调整图像大小。

```
In [16]: from PIL import Image
In [17]: img = Image.open('../_static/stinkbug.png')
In [18]: img.thumbnail((64, 64), Image.ANTIALIAS) # resizes imag
e in-place
In [19]: imgplot = plt.imshow(img)
```

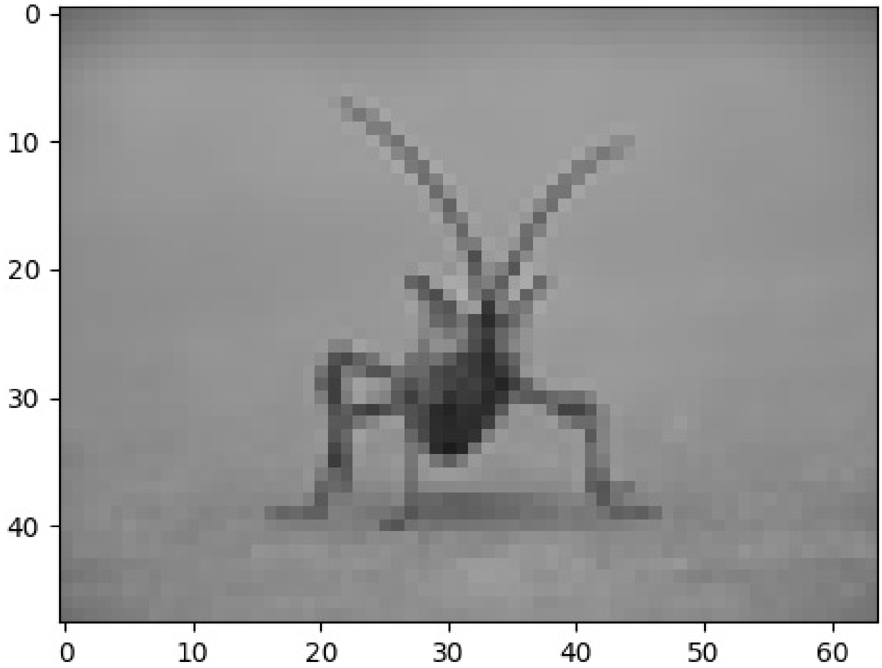


这里我们使用默认插值，双线性，因为我们没有向 `imshow()` 提供任何插值参数。

让我们试试一些其它的东西：

最邻近

```
In [20]: imgplot = plt.imshow(img, interpolation="nearest")
```



双立方

```
In [21]: imgplot = plt.imshow(img, interpolation="bicubic")
```



双立方插值通常用于放大照片 - 人们倾向于模糊而不是过度像素化。

使用 GridSpec 自定义子图位置

> 原文：Customizing Location of Subplot Using GridSpec
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

`GridSpec`

指定子图将放置的网格的几何位置。 需要设置网格的行数和列数。 子图布局参数（例如，左，右等）可以选择性调整。

`SubplotSpec`

指定在给定`GridSpec`中的子图位置。

`subplot2grid`

一个辅助函数，类似于`pyplot.subplot`，但是使用基于 0 的索引，并可使子图跨越多个格子。

`subplot2grid`基本示例

要使用subplot2grid，你需要提供网格的几何形状和网格中子图的位置。 对于简单的单网格子图：

```
ax = plt.subplot2grid((2,2),(0, 0))
```

等价于：

```
ax = plt.subplot(2,2,1)
```

```
      nRow=2, nCol=2
(0,0) +-------+-------+
      |   1   |       |
      +-------+-------+
      |       |       |
      +-------+-------+
```

要注意不想`subplot`，`gridspec`中的下标从 0 开始。

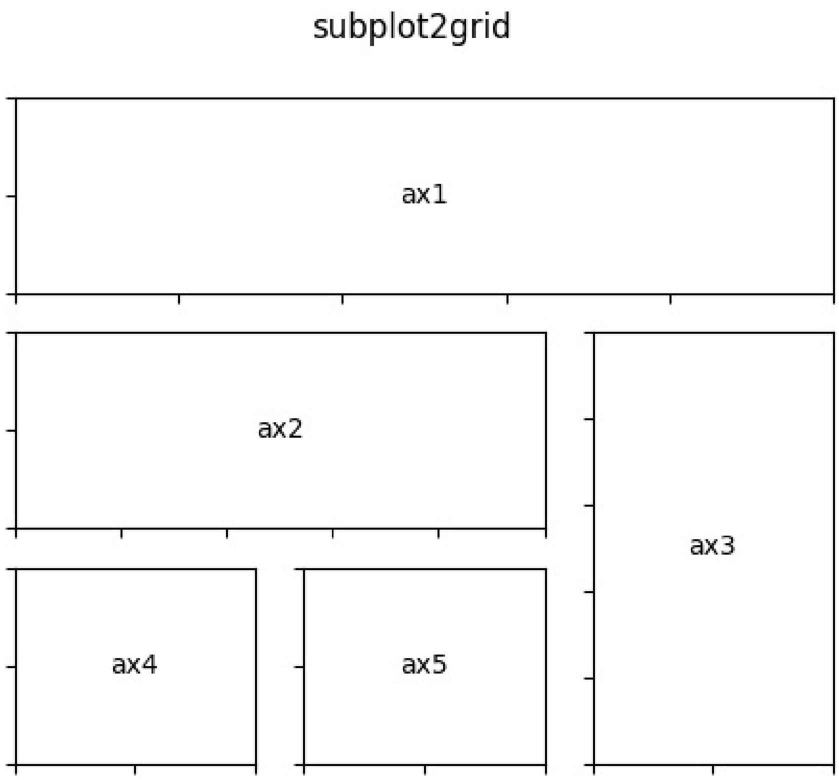
为了创建跨越多个格子的子图，

```
ax2 = plt.subplot2grid((3,3), (1, 0), colspan=2)
ax3 = plt.subplot2grid((3,3), (1, 2), rowspan=2)
```

例如，下列命令：

```
ax1 = plt.subplot2grid((3,3), (0,0), colspan=3)
ax2 = plt.subplot2grid((3,3), (1,0), colspan=2)
ax3 = plt.subplot2grid((3,3), (1, 2), rowspan=2)
ax4 = plt.subplot2grid((3,3), (2, 0))
ax5 = plt.subplot2grid((3,3), (2, 1))
```

会创建：



`GridSpec` 和 `SubplotSpec`

你可以显式创建 `GridSpec` 并用它们创建子图。

例如，

```
ax = plt.subplot2grid((2,2),(0, 0))
```

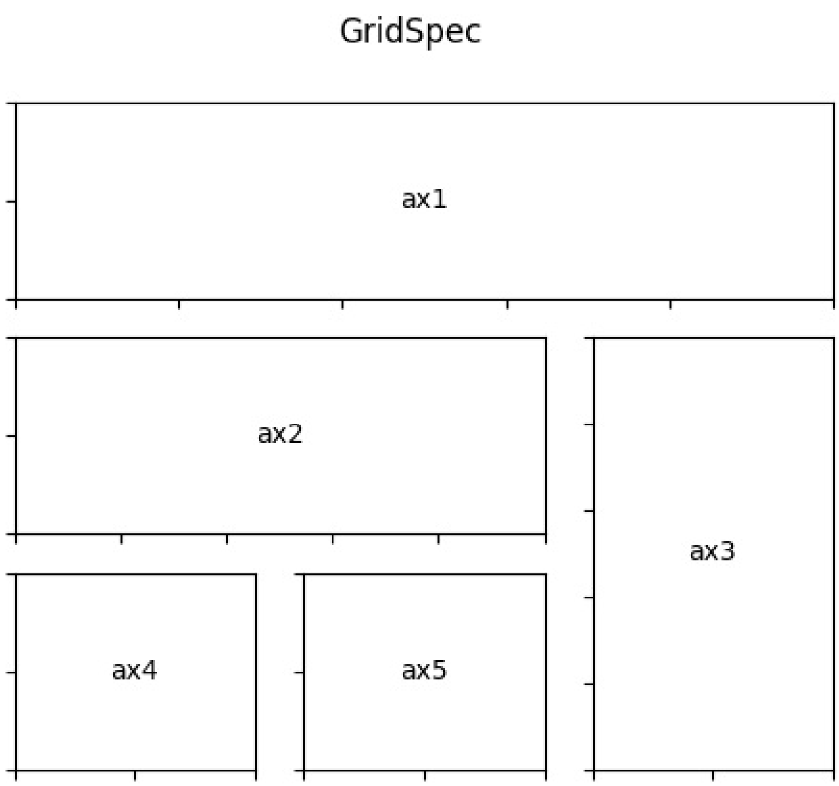
等价于：

```
import matplotlib.gridspec as gridspec
gs = gridspec.GridSpec(2, 2)
ax = plt.subplot(gs[0, 0])
```

`gridspec` 示例提供类似数组（一维或二维）的索引，并返回 `SubplotSpec` 实例。例如，使用切片来返回跨越多个格子的 `SubplotSpec` 实例。

上面的例子会变成：

```
gs = gridspec.GridSpec(3, 3)
ax1 = plt.subplot(gs[0, :])
ax2 = plt.subplot(gs[1,:-1])
ax3 = plt.subplot(gs[1:, -1])
ax4 = plt.subplot(gs[-1,0])
ax5 = plt.subplot(gs[-1,-2])
```



调整 `GridSpec` 布局

在显式使用 `GridSpec` 的时候，你可以调整子图的布局参数，子图由 `gridspec` 创建。

```
gs1 = gridspec.GridSpec(3, 3)
gs1.update(left=0.05, right=0.48, wspace=0.05)
```

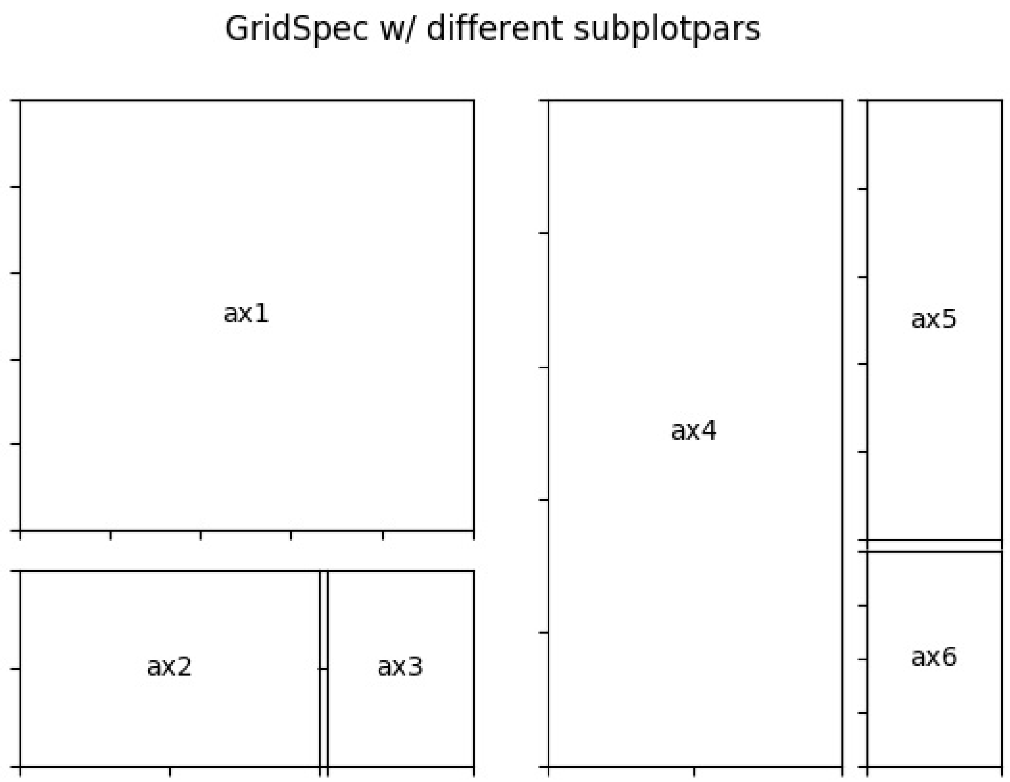
这类似于 `subplots_adjust` ，但是他只影响从给定 `GridSpec` 创建的子图。

下面的代码

```
gs1 = gridspec.GridSpec(3, 3)
gs1.update(left=0.05, right=0.48, wspace=0.05)
ax1 = plt.subplot(gs1[:-1, :])
ax2 = plt.subplot(gs1[-1, :-1])
ax3 = plt.subplot(gs1[-1, -1])

gs2 = gridspec.GridSpec(3, 3)
gs2.update(left=0.55, right=0.98, hspace=0.05)
ax4 = plt.subplot(gs2[:, :-1])
ax5 = plt.subplot(gs2[:-1, -1])
ax6 = plt.subplot(gs2[-1, -1])
```

会产生

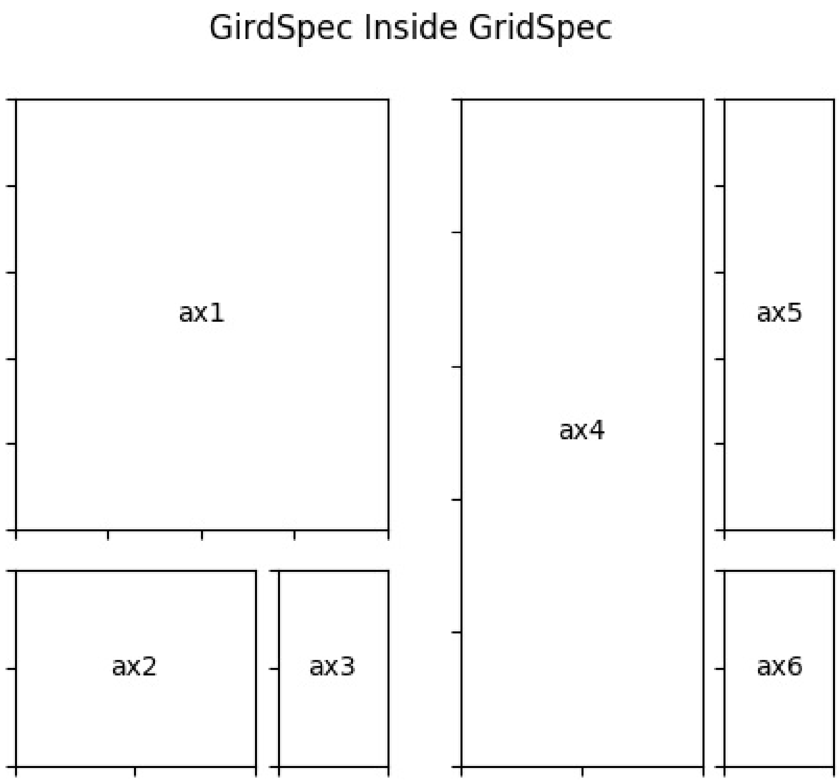


使用 `SubplotSpec` 创建 `GridSpec`

你可以从 `SubplotSpec` 创建 `GridSpec`，其中它的布局参数设置为给定 `SubplotSpec` 的位置的布局参数。

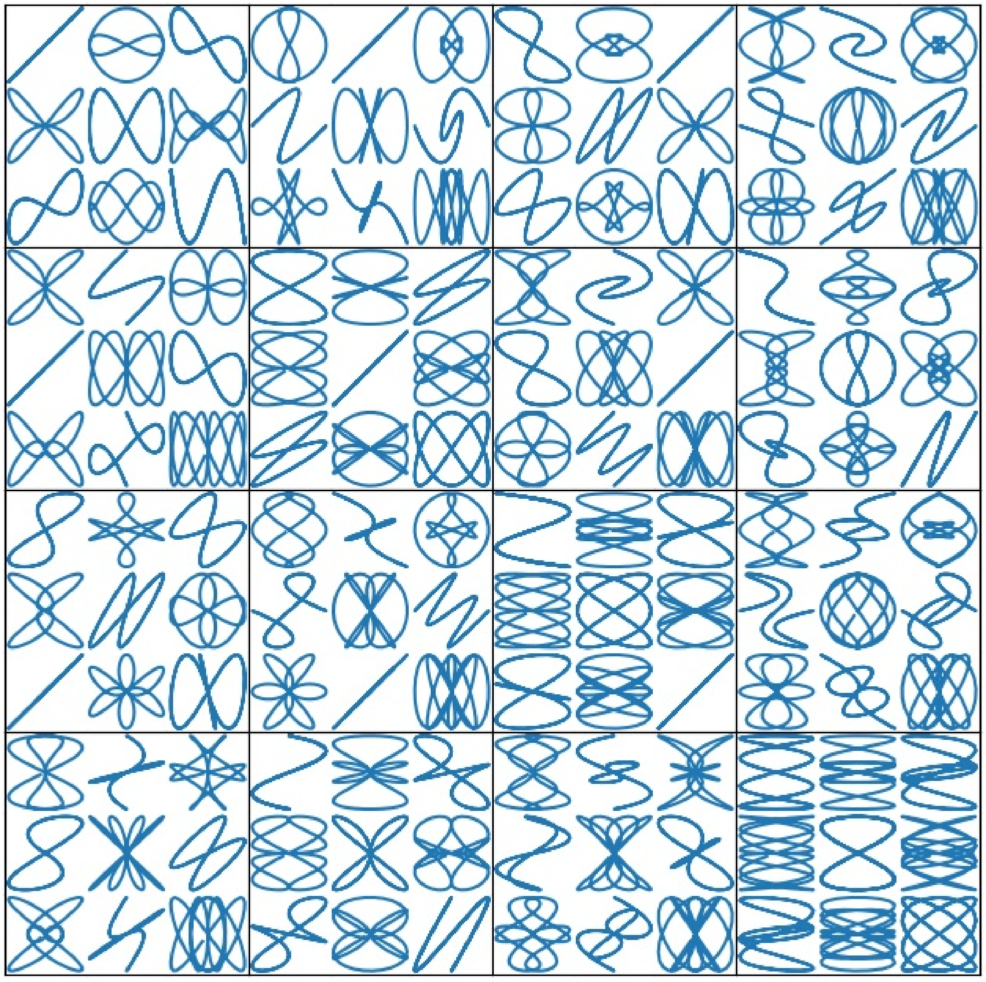
```
gs0 = gridspec.GridSpec(1, 2)

gs00 = gridspec.GridSpecFromSubplotSpec(3, 3, subplot_spec=gs0[0
])
gs01 = gridspec.GridSpecFromSubplotSpec(3, 3, subplot_spec=gs0[1
])
```



使用 `SubplotSpec` 创建复杂嵌套的 `GridSpec`

这里有一个更复杂的嵌套 `gridspec` 的示例，我们通过在每个 3x3 内部网格中隐藏适当的脊线，在 4x4 外部网格的每个单元格周围放置一个框。

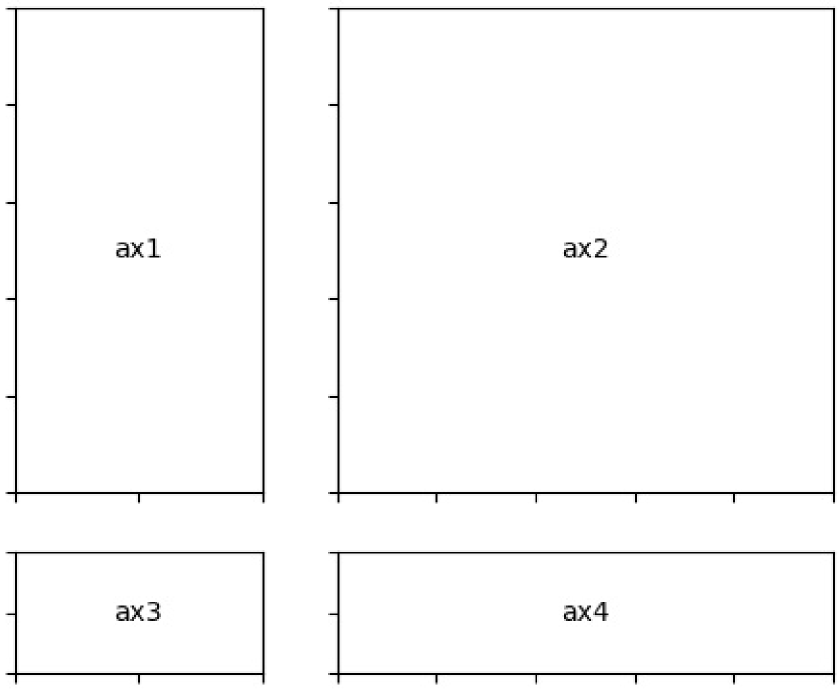


网格尺寸可变的 `GridSpec`

通常， `GridSpec` 创建大小相等的网格。你可以调整行和列的相对高度和宽度，要注意绝对高度值是无意义的，有意义的只是它们的相对比值。

```
gs = gridspec.GridSpec(2, 2,
                       width_ratios=[1,2],
                       height_ratios=[4,1]
                       )

ax1 = plt.subplot(gs[0])
ax2 = plt.subplot(gs[1])
ax3 = plt.subplot(gs[2])
ax4 = plt.subplot(gs[3])
```


ax1
ax2
ax3
ax4

密致布局指南

> 原文：Tight Layout guide
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

`tight_layout` 会自动调整子图参数，使之填充整个图像区域。这是个实验特性，可能在一些情况下不工作。它仅仅检查坐标轴标签、刻度标签以及标题的部分。

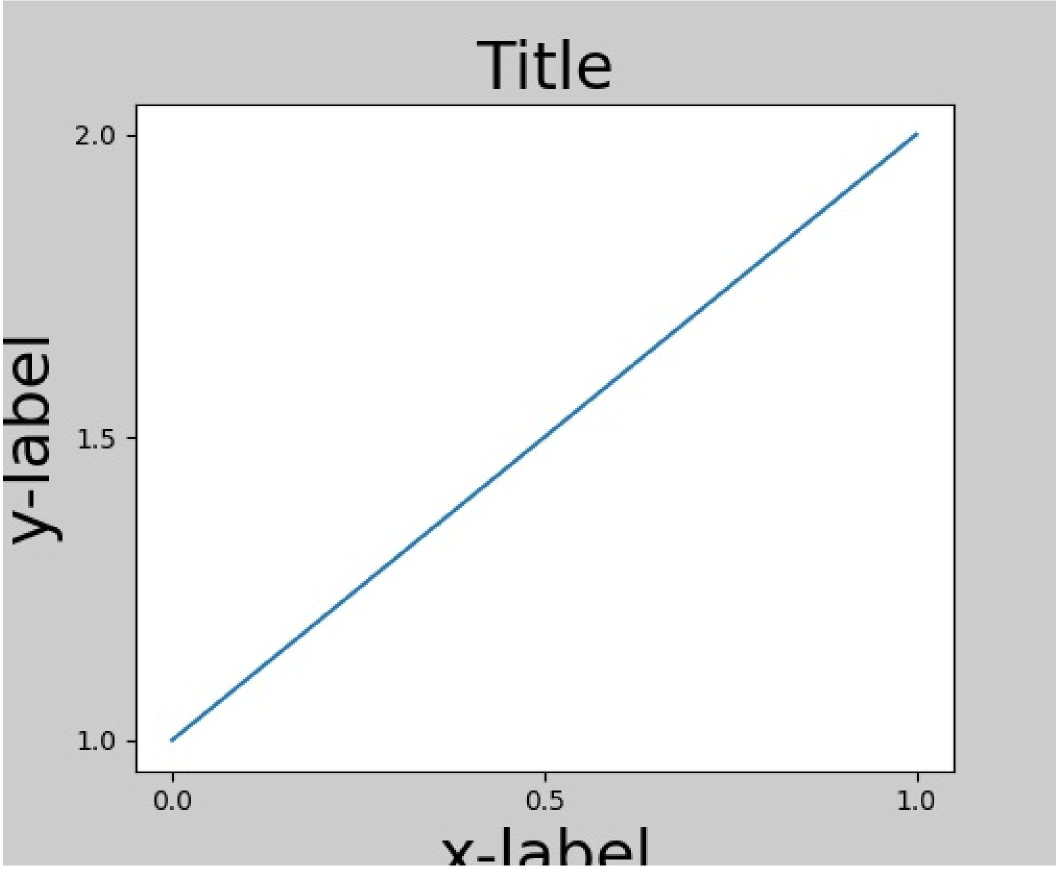
简单的示例

在 matplotlib 中，轴域（包括子图）的位置以标准化图形坐标指定。 可能发生的是，你的轴标签或标题（有时甚至是刻度标签）会超出图形区域，因此被截断。

```
plt.rcParams['savefig.facecolor'] = "0.8"

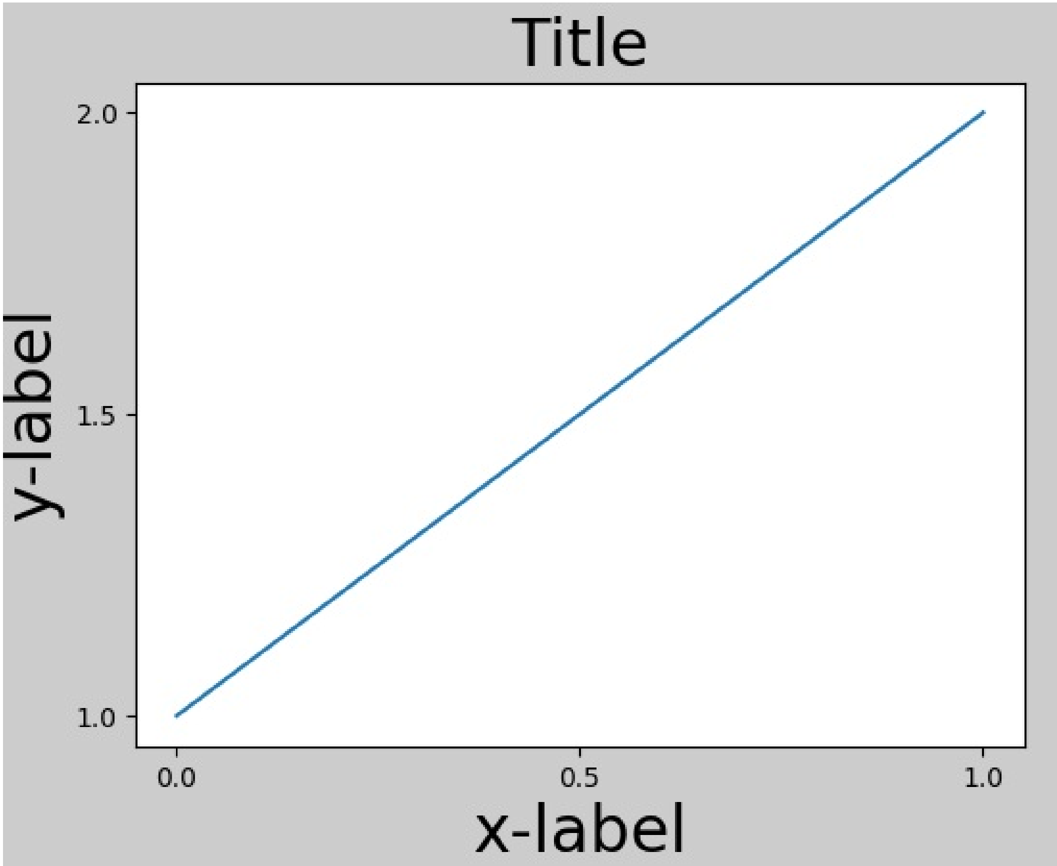
def example_plot(ax, fontsize=12):
     ax.plot([1, 2])
     ax.locator_params(nbins=3)
     ax.set_xlabel('x-label', fontsize=fontsize)
     ax.set_ylabel('y-label', fontsize=fontsize)
     ax.set_title('Title', fontsize=fontsize)

plt.close('all')
fig, ax = plt.subplots()
example_plot(ax, fontsize=24)
```



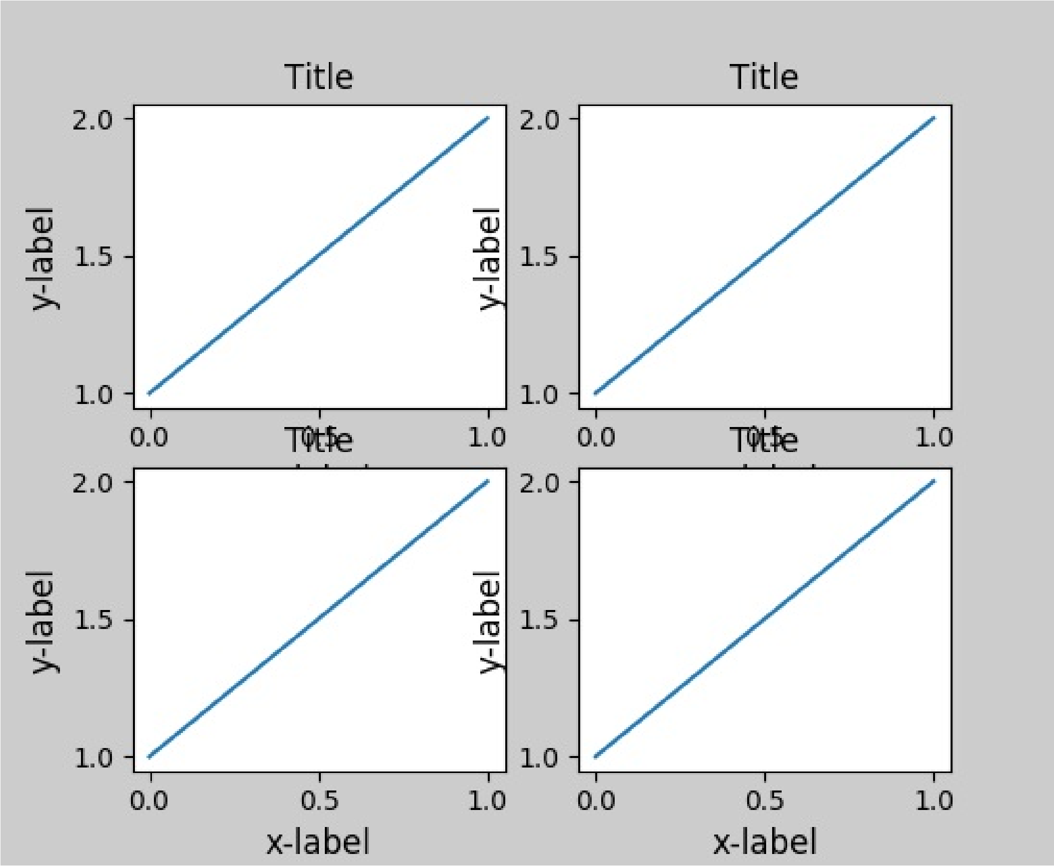
为了避免它，轴域的位置需要调整。对于子图，这可以通过调整子图参数（移动轴域的一条边来给刻度标签腾地方）。Matplotlib v1.1 引入了一个新的命令 `tight_layout()` ，自动为你解决这个问题。

```
plt.tight_layout()
```

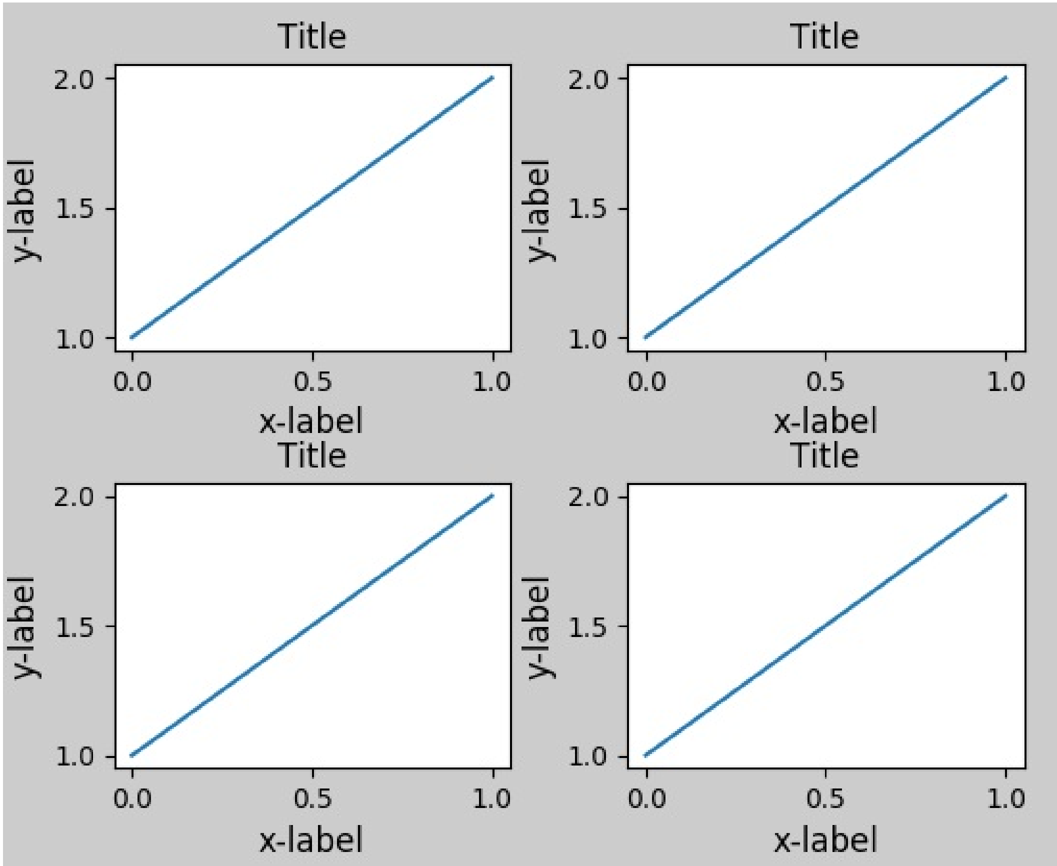


当你拥有多个子图时，你会经常看到不同轴域的标签叠在一起。

```
plt.close('all')
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows=2, ncols=2)
example_plot(ax1)
example_plot(ax2)
example_plot(ax3)
example_plot(ax4)
```

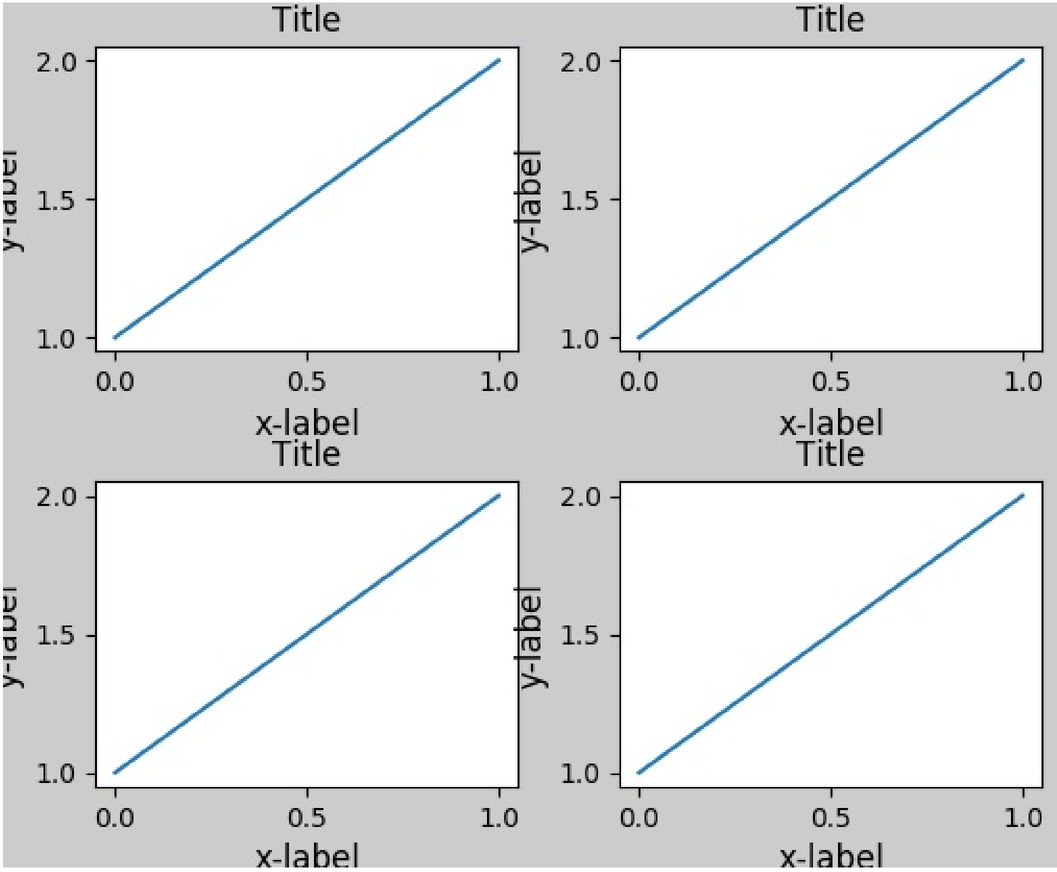


`tight_layout()` 也会调整子图之间的间隔来减少堆叠。



`tight_layout()` 可以接受关键字参数 `pad`、`w_pad` 或者 `h_pad`，这些参数图像边界和子图之间的额外边距。边距以字体大小单位规定。

```
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)
```



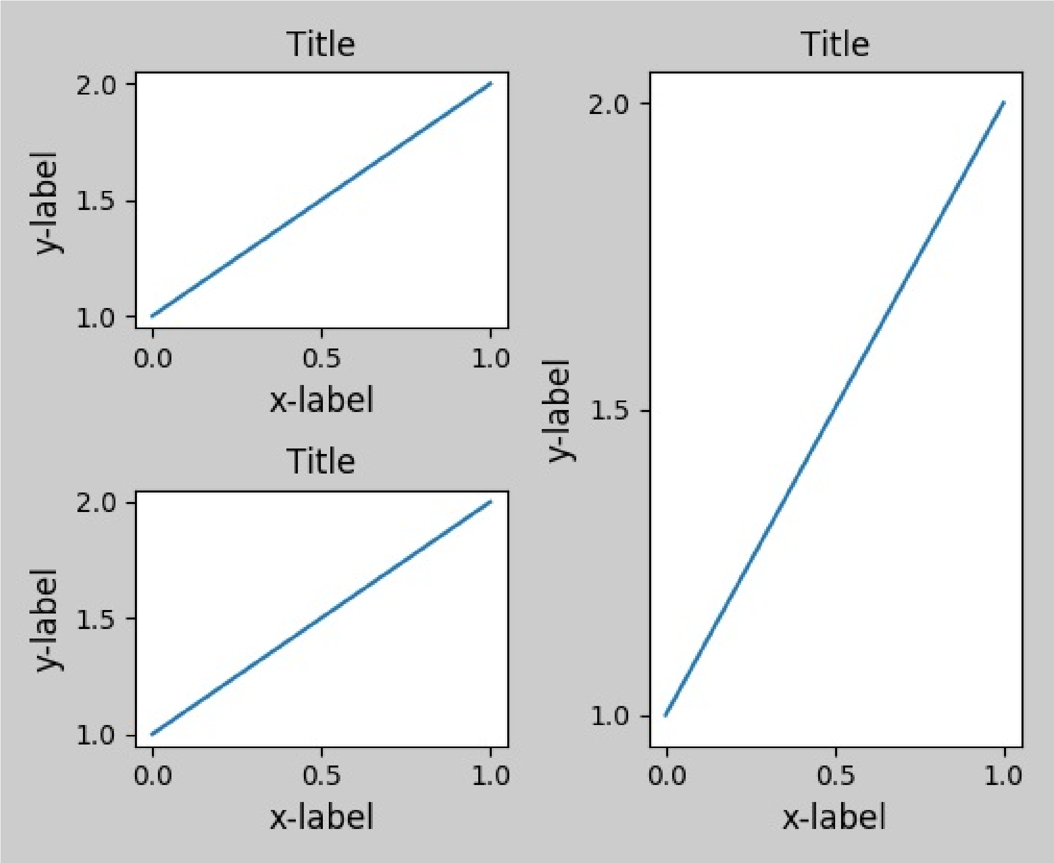
即使子图大小不同，`tight_layout()` 也能够工作，只要网格的规定的兼容的。在下面的例子中，`ax1` 和 `ax2` 是 2x2 网格的子图，但是 `ax3` 是 1x2 网格。

```
plt.close('all')
fig = plt.figure()

ax1 = plt.subplot(221)
ax2 = plt.subplot(223)
ax3 = plt.subplot(122)

example_plot(ax1)
example_plot(ax2)
example_plot(ax3)

plt.tight_layout()
```



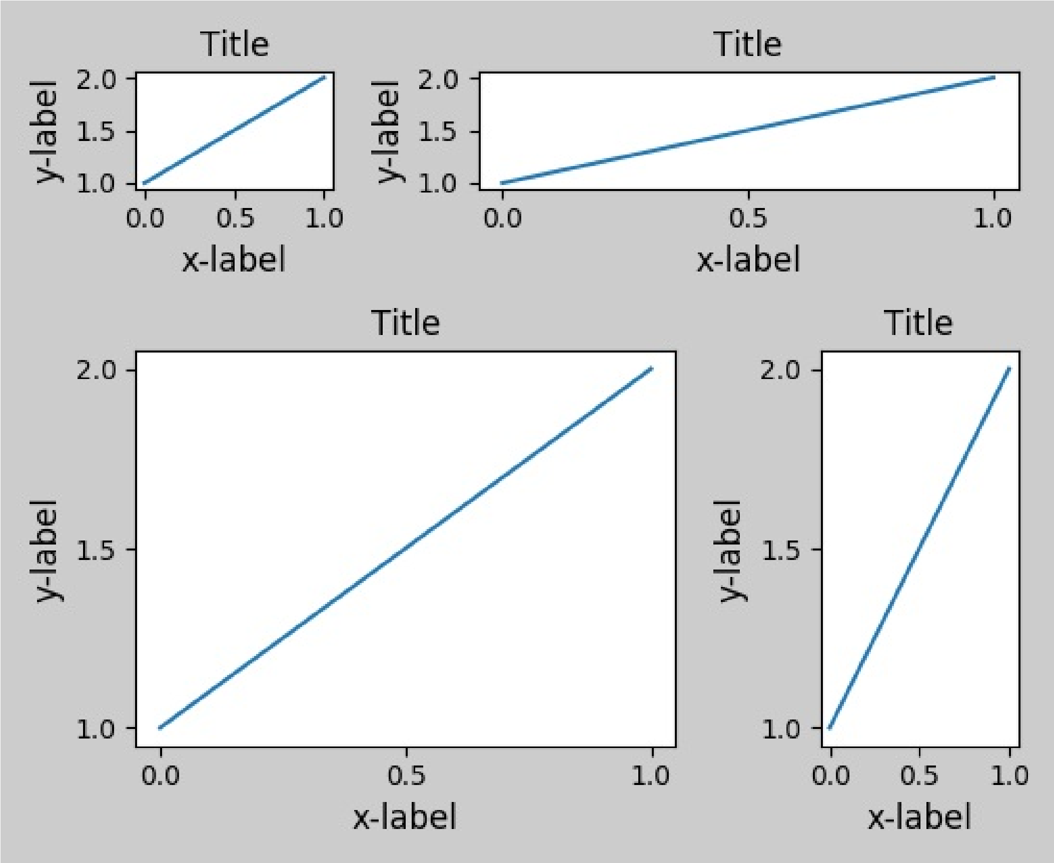
它适用于使用 `subplot2grid()` 创建的子图。一般来说，从 `gridspec`（使用 `GridSpec` 自定义子布局的位置）创建的子图也能正常工作。

```
plt.close('all')
fig = plt.figure()

ax1 = plt.subplot2grid((3, 3), (0, 0))
ax2 = plt.subplot2grid((3, 3), (0, 1), colspan=2)
ax3 = plt.subplot2grid((3, 3), (1, 0), colspan=2, rowspan=2)
ax4 = plt.subplot2grid((3, 3), (1, 2), rowspan=2)

example_plot(ax1)
example_plot(ax2)
example_plot(ax3)
example_plot(ax4)

plt.tight_layout()
```



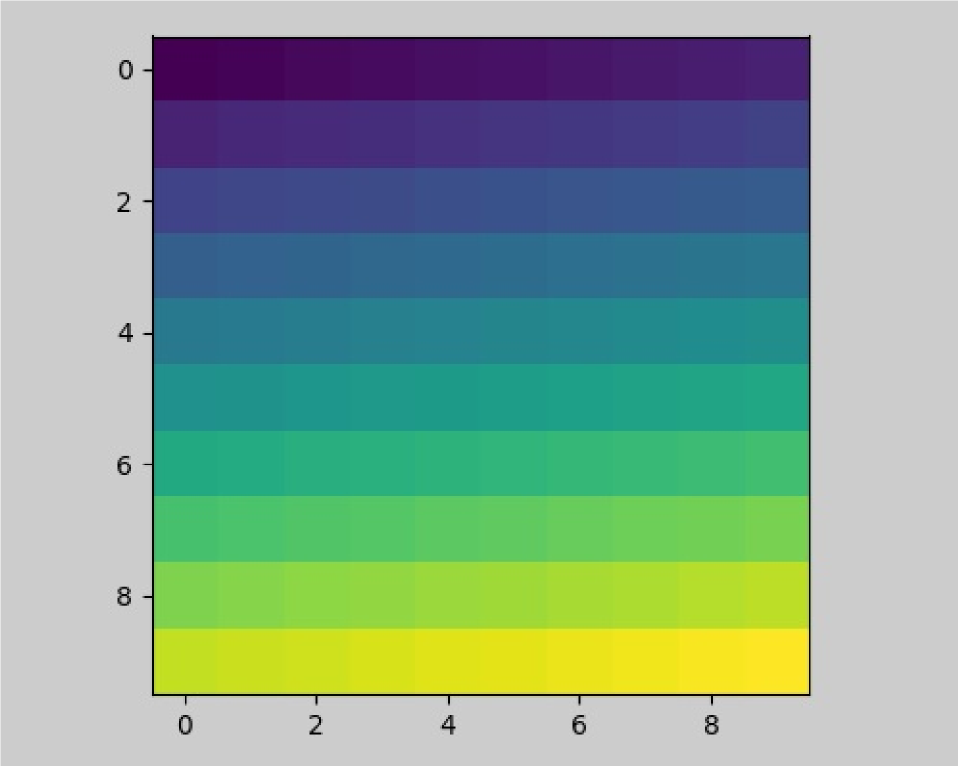
虽然没有彻底测试，它看起来也适用于 `aspect` 不为 `auto` 的子图（例如带有图像的轴域）。

```
arr = np.arange(100).reshape((10,10))

plt.close('all')
fig = plt.figure(figsize=(5,4))

ax = plt.subplot(111)
im = ax.imshow(arr, interpolation="none")

plt.tight_layout()
```



警告

- `tight_layout()` 只考虑刻度标签，轴标签和标题。 因此，其他艺术家可能被截断并且也可能重叠。
- 它假定刻度标签，轴标签和标题所需的额外空间与轴域的原始位置无关。 这通常是真的，但在罕见的情况下不是。
- `pad = 0` 将某些文本剪切几个像素。 这可能是当前算法的错误或限制，并且不清楚为什么会发生。 同时，推荐使用至少大于 0.3 的间隔。

和 `GridSpec` 一起使用

`GridSpec` 拥有自己的 `tight_layout()` 方法（pyplot API 的 `tight_layout()` 也能生效）。

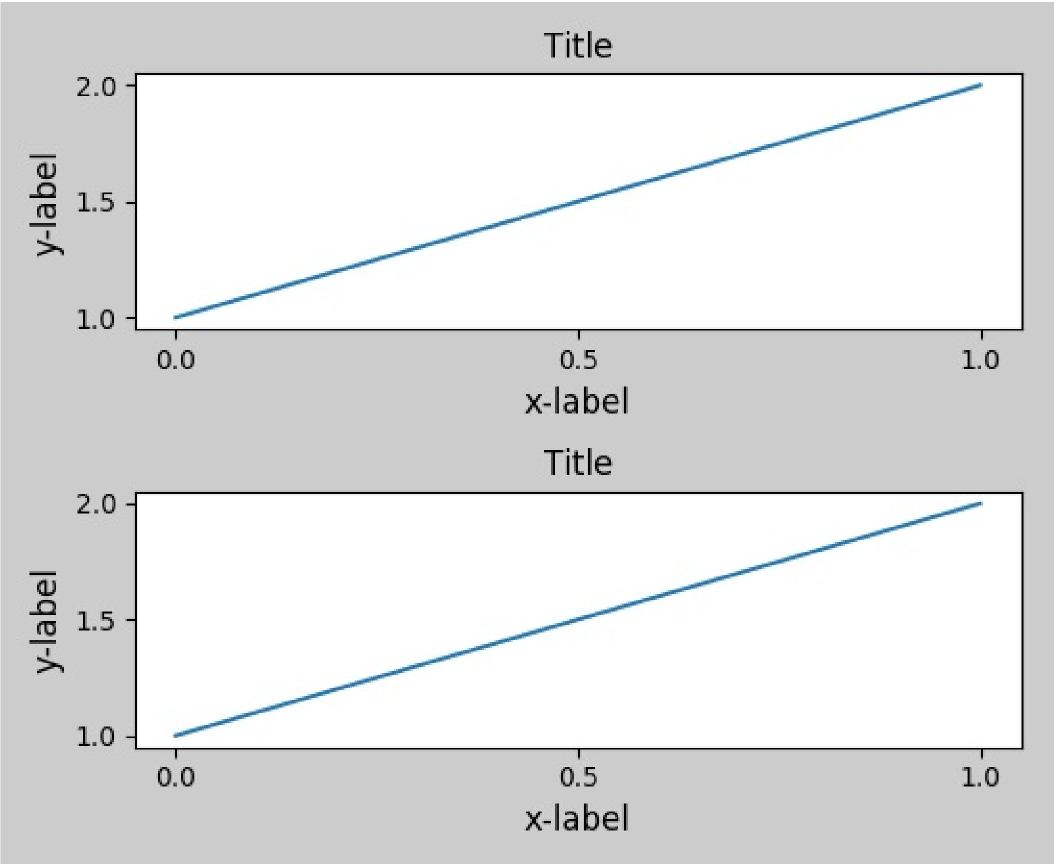
```
plt.close('all')
fig = plt.figure()

import matplotlib.gridspec as gridspec

gs1 = gridspec.GridSpec(2, 1)
ax1 = fig.add_subplot(gs1[0])
ax2 = fig.add_subplot(gs1[1])

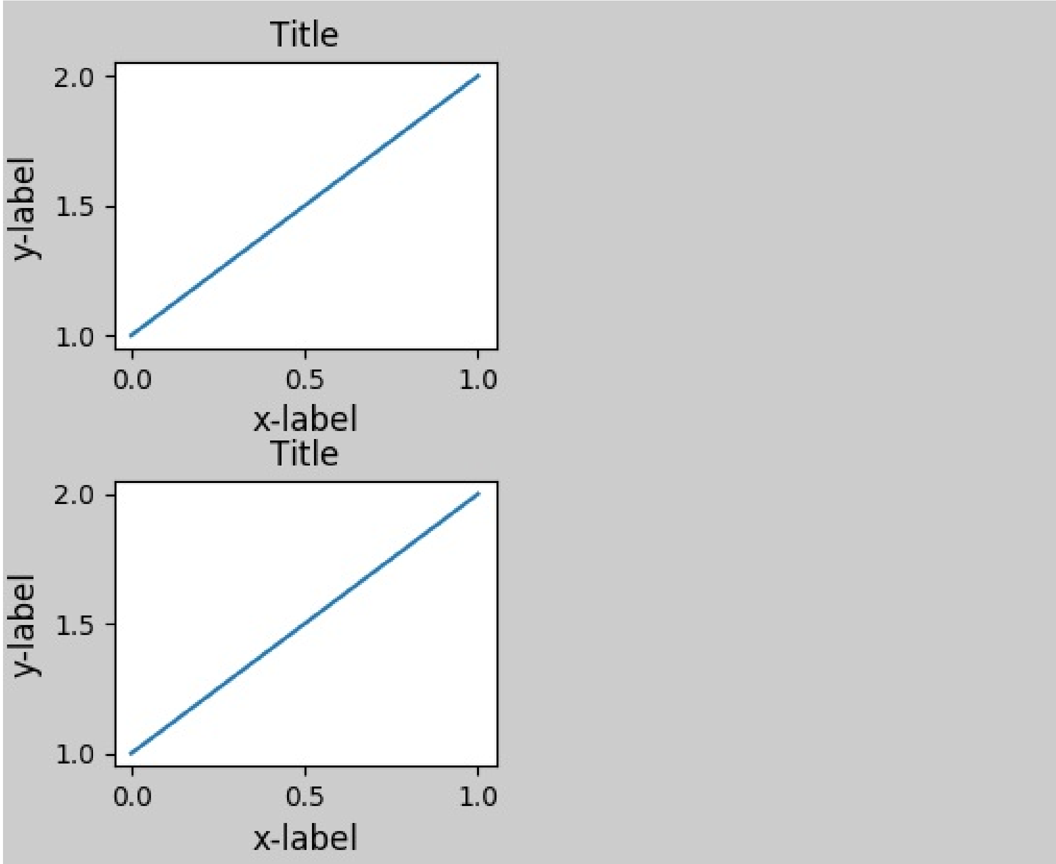
example_plot(ax1)
example_plot(ax2)

gs1.tight_layout(fig)
```



你可以提供一个可选的 `rect` 参数，指定子图所填充的边框。 坐标必须为标准化图形坐标，默认值为 `(0, 0, 1, 1)` 。

```
gs1.tight_layout(fig, rect=[0, 0, 0.5, 1])
```



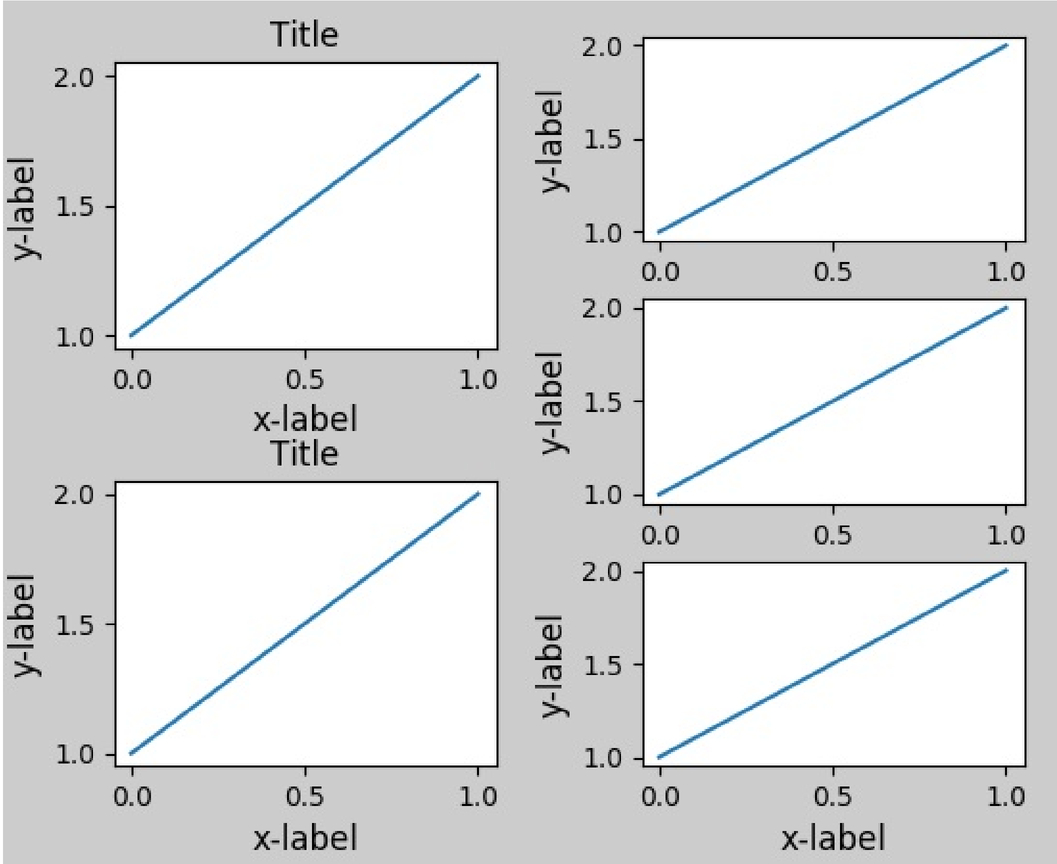
例如，这可用于带有多个 `gridspecs` 的图形。

```
gs2 = gridspec.GridSpec(3, 1)

for ss in gs2:
    ax = fig.add_subplot(ss)
    example_plot(ax)
    ax.set_title("")
    ax.set_xlabel("")

ax.set_xlabel("x-label", fontsize=12)

gs2.tight_layout(fig, rect=[0.5, 0, 1, 1], h_pad=0.5)
```



我们可以尝试匹配两个网格的顶部和底部。

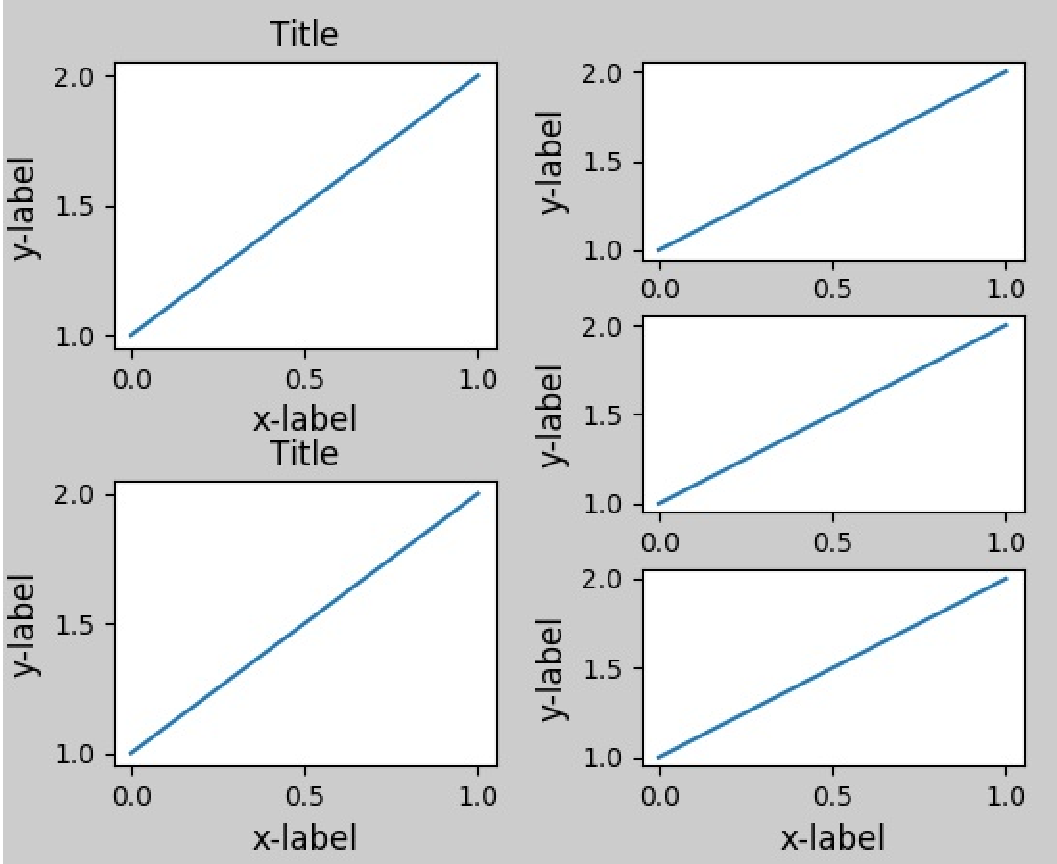
```
top = min(gs1.top, gs2.top)
bottom = max(gs1.bottom, gs2.bottom)

gs1.update(top=top, bottom=bottom)
gs2.update(top=top, bottom=bottom)
```

虽然这应该足够好了，调整顶部和底部可能也需要调整 `hspace` 。 为了更新 `hspace` 和 `vspace` ，我们再次使用更新后的 `rect` 参数调用 `tight_layout()` 。 注意， `rect` 参数指定的区域包括刻度标签。因此，我们将底部（正常情况下为 0）增加每个 `gridspec` 的底部之差。 顶部也一样。

```
top = min(gs1.top, gs2.top)
bottom = max(gs1.bottom, gs2.bottom)

gs1.tight_layout(fig, rect=[None, 0 + (bottom-gs1.bottom),
                            0.5, 1 - (gs1.top-top)])
gs2.tight_layout(fig, rect=[0.5, 0 + (bottom-gs2.bottom),
                            None, 1 - (gs2.top-top)],
                 h_pad=0.5)
```



和 `AxesGrid1` 一起使用

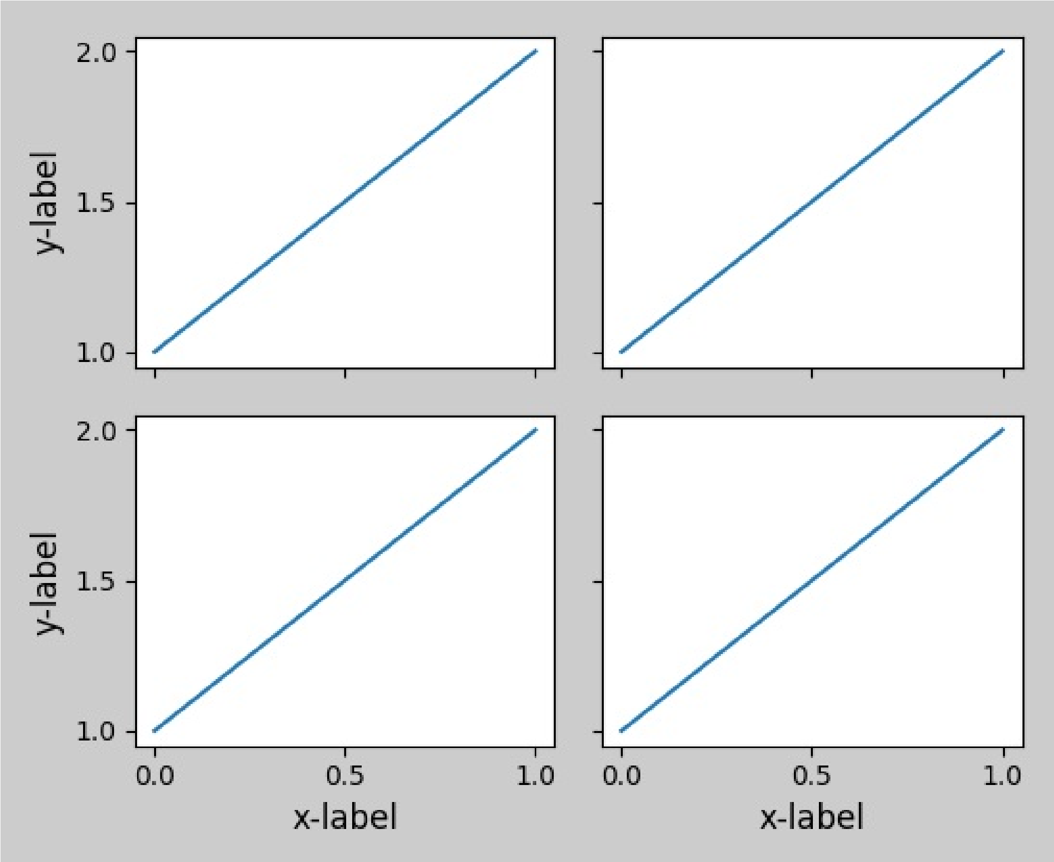
虽然受限但也支持 `axes_grid1` 工具包

```
plt.close('all')
fig = plt.figure()

from mpl_toolkits.axes_grid1 import Grid
grid = Grid(fig, rect=111, nrows_ncols=(2,2),
            axes_pad=0.25, label_mode='L',
            )

for ax in grid:
    example_plot(ax)
    ax.title.set_visible(False)

plt.tight_layout()
```



颜色条

如果你使用 `colorbar` 命令创建了颜色条，创建的颜色条是 `Axes` 而不是 `Subplot` 的实例，所以 `tight_layout` 没有效果。在 Matplotlib v1.1 中，你可以使用 `gridspec` 将颜色条创建为子图。

```
plt.close('all')
arr = np.arange(100).reshape((10,10))
fig = plt.figure(figsize=(4, 4))
im = plt.imshow(arr, interpolation="none")

plt.colorbar(im, use_gridspec=True)

plt.tight_layout()
```

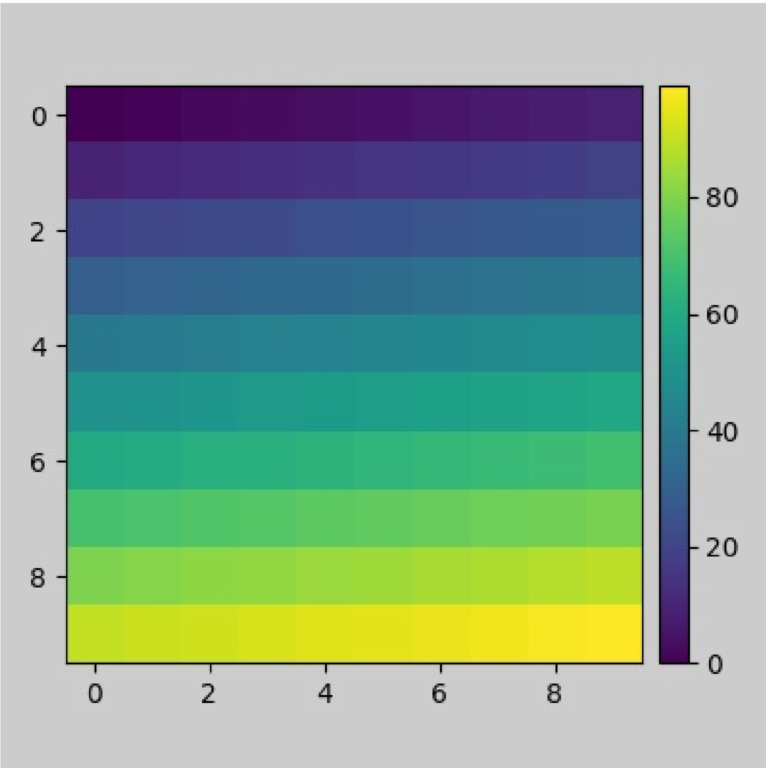
![])http://matplotlib.org/_images/tight_layout_guide-14.png

另一个选项是使用 `AxesGrid1` 工具包，显式为颜色条创建一个轴域：

```
plt.close('all')
arr = np.arange(100).reshape((10,10))
fig = plt.figure(figsize=(4, 4))
im = plt.imshow(arr, interpolation="none")

from mpl_toolkits.axes_grid1 import make_axes_locatable
divider = make_axes_locatable(plt.gca())
cax = divider.append_axes("right", "5%", pad="3%")
plt.colorbar(im, cax=cax)

plt.tight_layout()
```



艺术家教程

> 原文：[Artist tutorial]
>
> 译者：[飞龙]
>
> 协议：[CC BY-NC-SA 4.0]

matplotlib API 有三个层级。 `matplotlib.backend_bases.FigureCanvas` 是绘制图形的区域， `matplotlib.backend_bases.Renderer` 是知道如何在 `ChartCanvas` 上绘制的对象，而 `matplotlib.artist.Artist` 是知道如何使用渲染器在画布上画图的对象。 `FigureCanvas` 和 `Renderer` 处理与用户界面工具包（如 wxPython）或 PostScript® 等绘图语言交互的所有细节， `Artist` 处理所有高级结构，如表示和布局图形，文本和线条。用户通常要花费95％的时间来处理艺术家。

有两种类型的艺术家：基本类型和容器类型。基本类型表示我们想要绘制到画布上的标准图形对象： `Line2D` ， `Rectangle` ， `Text` ， `AxesImage` 等，容器是放置它们的位置（ `Axis` ， `Axes` 和 `Figure` ）。标准用法是创建一个 `Figure` 实例，使用 `Figure` 创建一个或多个 `Axes` 或 `Subplot` 实例，并使用 `Axes` 实例的辅助方法来创建基本类型。在下面的示例中，我们使用 `matplotlib.pyplot.figure()` 创建一个 `Figure` 实例，这是一个便捷的方法，用于实例化 `Figure` 实例并将它们与你的用户界面或绘图工具包 `FigureCanvas` 连接。正如我们将在下面讨论的，这不是必须的 - 你可以直接使用 PostScript，PDF，Gtk+ 或 wxPython `FigureCanvas` 实例，直接实例化你的图形并连接它们 - 但是因为我们在这里关注艺术家 API，我们让 `pyplot` 为我们处理一些细节：

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(2,1,1) # two rows, one column, first plot
```

`Axes` 可能是 matplotlib API 中最重要的类，你将在大多数时间使用它。 这是因为 `Axes` 是大多数对象所进入的绘图区域， `Axes` 有许多特殊的辅助方法（ `plot()` ， `text()` ， `hist()` ， `imshow()` ）来创建最常见的图形基本类型 `Line2D` ， `Text` ， `Rectangle` ， `Image` ）。 这些辅助方法将获取你的数据（例如 numpy 数组和字符串），并根据需要创建基本 `Artist` 实例（例如， `Line2D` ），将它们添加到相关容器中，并在请求时绘制它们。 大多数人可能熟悉子图，这只是 `Axes` 的一个特例，它存在于 `Subplot` 实例的列网格的固定行上。 如果要在任意位置创建 `Axes` ，只需使用 `add_axes()` 方法，该方法接受 `[left, bottom, width, height]` 值的列表，以 0~1 的图形相对坐标为单位：

```
fig2 = plt.figure()
ax2 = fig2.add_axes([0.15, 0.1, 0.7, 0.3])
```

以我们的例子继续：

```
import numpy as np
t = np.arange(0.0, 1.0, 0.01)
s = np.sin(2*np.pi*t)
line, = ax.plot(t, s, color='blue', lw=2)
```

在这个例子中， `ax` 是上面的 `fig.add_subplot` 调用创建的 `Axes` 实例（记住 `Subplot` 只是 `Axes` 的一个子类），当你调用 `ax.plot` 时，它创建一个 `Line2D` 实例并将其添加到 `Axes.lines` 列表中。 在下面的 ipython 交互式会话中，你可以看到 `Axes.lines` 列表的长度为 1，并且包含由 `line, = ax.plot...` 调用返回的相同线条：

```
In [101]: ax.lines[0]
Out[101]: <matplotlib.lines.Line2D instance at 0x19a95710>

In [102]: line
Out[102]: <matplotlib.lines.Line2D instance at 0x19a95710>
```

如果你对 `ax.plot` 进行连续调用（并且保持状态为『on』，这是默认值），则将在列表中添加其他线条。 你可以稍后通过调用列表方法删除线条；任何一个方法都可以：

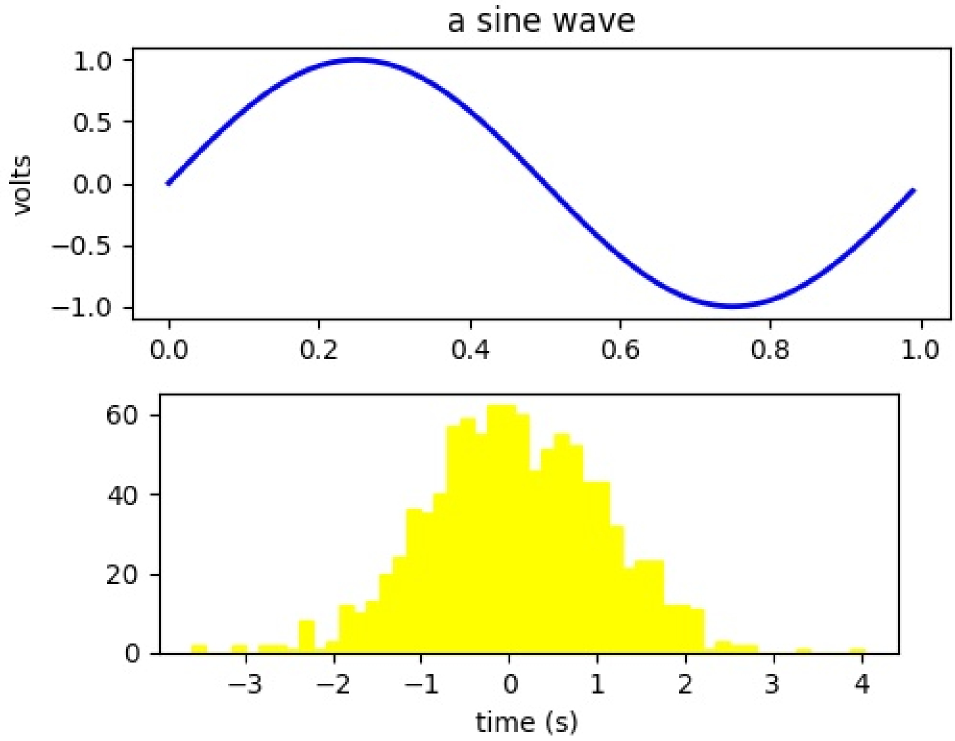
```
del ax.lines[0]
ax.lines.remove(line)  # one or the other, not both!
```

轴域也拥有辅助方法，用于设置和装饰 x 和 y 轴的刻度、刻度标签和轴标签：

```
xtext = ax.set_xlabel('my xdata') # returns a Text instance
ytext = ax.set_ylabel('my ydata')
```

当你调用 `ax.set_xlabel` 时，它将信息传递给 `XAxis` 的 `Text` 实例，每个 `Axes` 实例都包含 `XAxis` 和 `YAxis` ，它们处理刻度、刻度标签和轴标签的布局和绘制。

尝试创建下面的图形：



自定义你的对象

图中的每个元素都由一个 matplotlib 艺术家表示，每个元素都有一个扩展属性列表用于配置它的外观。图形本身包含一个 `Rectangle` ，正好是图形的大小，你可以使用它来设置图形的背景颜色和透明度。同样，每个 `Axes` 边框（在通常的 matplotlib 绘图中是标准的白底黑边）拥有一个 `Rectangle` 实例，用于确定轴域的颜色，透明度和其他属性，这些实例存储为成员变量 `Figure.patch` 和 `Axes.patch` （『Patch』是一个继承自 MATLAB 的名称，它是图形上的一个颜色的 2D『补丁』，例如矩形，圆和多边形）。每个 matplotlib 艺术家都有以下属性。

属性	描述
alpha	透明度 - 0 ~ 1 的标量
animated	用于帮助动画绘制的布尔值
axes	艺术家所在的轴域，可能为空
clip_box	用于剪切艺术家的边框
clip_on	剪切是否开启
clip_path	艺术家被剪切的路径
contains	一个拾取函数，用于判断艺术家是否位于拾取点
figure	艺术家所在的图形实例，可能为空
label	文本标签（用于自动标记）
picker	控制对象拾取的 Python 对象
transform	变换
visible	布尔值，表示艺术家是否应该绘制
zorder	确定绘制顺序的数值
rasterized	布尔值，是否将向量转换为光栅图形（出于压缩或 eps 透明度）

每个属性都使用一个老式的 `setter` 或 `getter` （是的，我们知道这会刺激 Python 爱好者，我们计划支持通过属性或 traits 直接访问，但它还没有完成）。 例如，要将当前 `alpha` 值变为一半：

```
a = o.get_alpha()
o.set_alpha(0.5*a)
```

如果你打算可以一次性设置一些属性，你也可以以关键字参数使用 `set` 方法，例如：

```
o.set(alpha=0.5, zorder=2)
```

如果你在 Python 交互式 Shell 中工作，检查 `Artist` 属性的一种方便的方法是使用 `matplotlib.artist.getp()` 函数（在 pylab 中只需要 `getp()` ），它列出了属性及其值。 这适用于从 `Artist` 派生的类，例如 `Figure` 和 `Rectangle` 。 这里是上面提到的 `Figure` 的矩形属性：

```
In [149]: matplotlib.artist.getp(fig.patch)
    alpha = 1.0
    animated = False
    antialiased or aa = True
    axes = None
    clip_box = None
    clip_on = False
    clip_path = None
    contains = None
    edgecolor or ec = w
    facecolor or fc = 0.75
    figure = Figure(8.125x6.125)
    fill = 1
    hatch = None
    height = 1
    label =
    linewidth or lw = 1.0
    picker = None
    transform = <Affine object at 0x134cca84>
    verts = ((0, 0), (0, 1), (1, 1), (1, 0))
    visible = True
    width = 1
    window_extent = <Bbox object at 0x134acbcc>
    x = 0
    y = 0
    zorder = 1
```

所有类的文档字符串也包含 `Artist` 属性，因此你可以查阅交互式『帮助』或 `Artist` 模块，来获取给定对象的属性列表。

对象容器

现在我们知道如何检查和设置我们想要配置的给定对象的属性，现在我们需要如何获取该对象。 前面提到了两种对象：基本类型和容器类型。 基本类型通常是你想要配置的东西（ `Text` 实例的字体， `Line2D` 的宽度），虽然容器也有一些属性 - 例如 `Axes` 是一个容器艺术家，包含你的绘图中的许多基本类型，但它也有属性，比如 `xscale` 来控制 `xaxis` 是『线性』还是『对数』。 在本节中，我们将回顾各种容器对象存储你想要访问的艺术家的位置。

图形容器

顶层容器艺术家是 `matplotlib.figure.Figure` ，它包含图形中的所有内容。 图形的背景是一个 `Rectangle` ，存储在 `Figure.patch` 中。 当你向图形中添加子图（ `add_subplot()` ）和轴域（ `add_axes()` ）时，这些会附加到 `Figure.axes` 。 它们也由创建它们的方法返回：

```
In [156]: fig = plt.figure()

In [157]: ax1 = fig.add_subplot(211)

In [158]: ax2 = fig.add_axes([0.1, 0.1, 0.7, 0.3])

In [159]: ax1
Out[159]: <matplotlib.axes.Subplot instance at 0xd54b26c>

In [160]: print fig.axes
[<matplotlib.axes.Subplot instance at 0xd54b26c>, <matplotlib.ax
es.Axes instance at 0xd3f0b2c>]
```

因为图形维护了『当前轴域』（见 `figure.gca` 和图 `figure.sca` ）的概念以支持 pylab/pyplot 状态机，所以不应直接从轴域列表中插入或删除轴域，而应使用 `add_subplot()` 和 `add_axes()` 方法进行插入，并使用 `delaxes()` 方法进行删除。 然而，你可以自由地遍历轴域列表或索引，来访问要自定义的 `Axes` 实例。 下面是一个打开所有轴域网格的示例：

```
for ax in fig.axes:
    ax.grid(True)
```

图形还拥有自己的文本，线条，补丁和图像，你可以使用它们直接添加基本类型。 图形的默认坐标系统简单地以像素（这通常不是你想要的）为单位，但你可以通过设置你添加到图中的艺术家的 `transform` 属性来控制它。

更有用的是『图形坐标系』，其中 `(0,0)` 是图的左下角， `(1,1)` 是图的右上角，你可以通过将 `Artist` 的变换设置为 `fig.transFigure` 来获得：

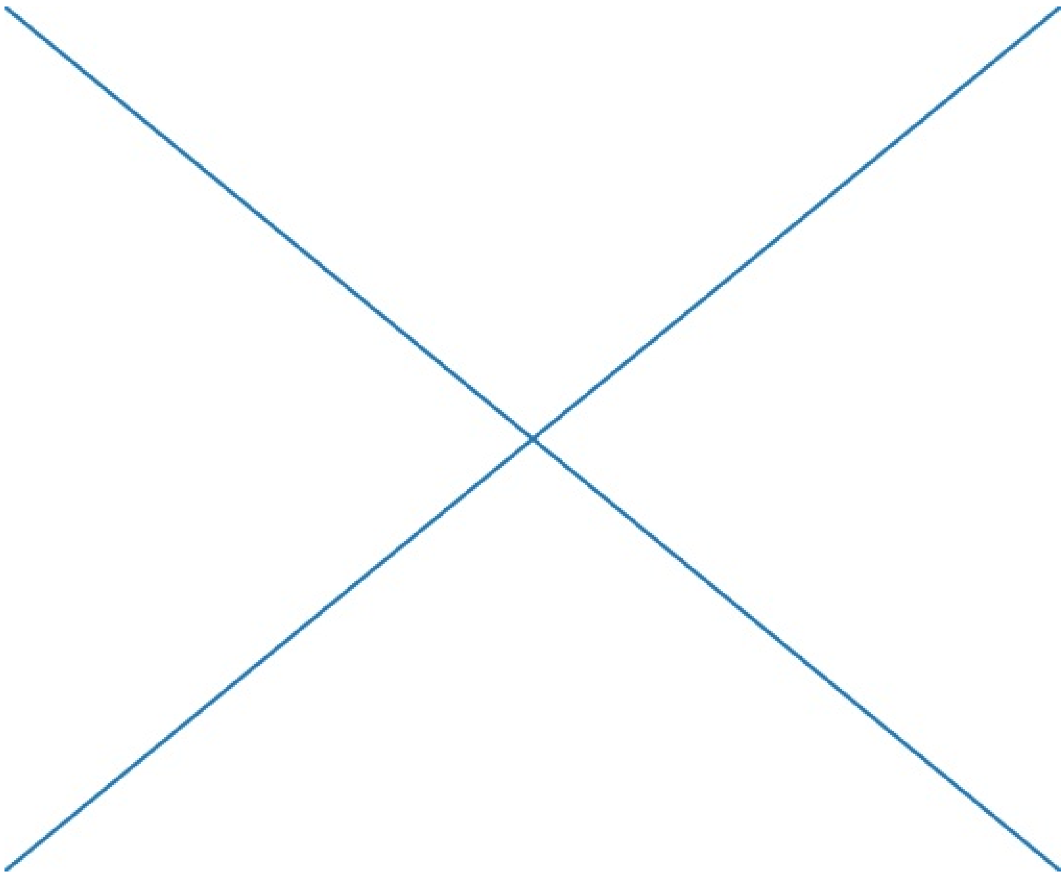
```
In [191]: fig = plt.figure()

In [192]: l1 = matplotlib.lines.Line2D([0, 1], [0, 1],
           transform=fig.transFigure, figure=fig)

In [193]: l2 = matplotlib.lines.Line2D([0, 1], [1, 0],
           transform=fig.transFigure, figure=fig)

In [194]: fig.lines.extend([l1, l2])

In [195]: fig.canvas.draw()
```



这里是图形可以包含的艺术家总结：

图形属性	描述
`axes`	`Axes` 实例的列表（包括 `Subplot` ）
`patch`	`Rectangle` 背景
`images`	`FigureImages` 补丁的列表 - 用于原始像素显示
`legends`	图形 `Legend` 实例的列表（不同于 `Axes.legends` ）
`lines`	图形 `Line2D` 实例的列表（很少使用，见 `Axes.lines` ）
`patches`	图形补丁列表（很少使用，见 `Axes.patches` ）
`texts`	图形 `Text` 实例的列表

轴域容器

`matplotlib.axes.Axes` 是 matplotlib 宇宙的中心 - 它包含绝大多数在一个图形中使用的艺术家，并带有许多辅助方法来创建和添加这些艺术家本身，以及访问和自定义所包含的艺术家的辅助方法。 就像 `Figure` 那样，它包含一个 `Patch patch` ，它是一个用于笛卡尔坐标的 `Rectangle` 和一个用于极坐标的 `Circle` ；这个补丁决定了绘图区域的形状，背景和边框：

```
ax = fig.add_subplot(111)
rect = ax.patch  # a Rectangle instance
rect.set_facecolor('green')
```

当调用绘图方法（例如通常是 `plot()` ）并传递数组或值列表时，该方法将创建一个 `matplotlib.lines.Line2D()` 实例，将所有 `Line2D` 属性作为关键字参数传递， 将该线条添加到 `Axes.lines` 容器，并将其返回给你：

```
In [213]: x, y = np.random.rand(2, 100)

In [214]: line, = ax.plot(x, y, '-', color='blue', linewidth=2)
```

`plot` 返回一个线条列表，因为你可以传入多个 `x,y` 偶对来绘制，我们将长度为一的列表的第一个元素解构到 `line` 变量中。 该线条已添加到 `Axes.lines` 列表中：

```
In [229]: print ax.lines
[<matplotlib.lines.Line2D instance at 0xd378b0c>]
```

与之类似，创建补丁的方法（如 `bar()` ）会创建一个矩形列表，将补丁添加到 `Axes.patches` 列表中：

```
In [233]: n, bins, rectangles = ax.hist(np.random.randn(1000), 50
, facecolor='yellow')

In [234]: rectangles
Out[234]: <a list of 50 Patch objects>

In [235]: print len(ax.patches)
```

你不应该直接将对象添加到 `Axes.lines` 或 `Axes.patches` 列表，除非你确切知道你在做什么，因为 `Axes` 需要在它创建和添加对象做一些事情。 它设置 `Artist` 的 `figure` 和 `axes` 属性，以及默认 `Axes` 变换（除非设置了变换）。 它还检查 `Artist` 中包含的数据，来更新控制自动缩放的数据结构，以便可以调整视图限制来包含绘制的数据。 但是，你可以自己创建对象，并使用辅助方法（如 `add_line()` 和 `add_patch()` ）将它们直接添加到 `Axes` 。 这里是一个注释的交互式会话，说明正在发生什么：

```
In [261]: fig = plt.figure()

In [262]: ax = fig.add_subplot(111)

# create a rectangle instance
In [263]: rect = matplotlib.patches.Rectangle( (1,1), width=5, h
eight=12)

# by default the axes instance is None
In [264]: print rect.get_axes()
None

# and the transformation instance is set to the "identity transf
orm"
In [265]: print rect.get_transform()
<Affine object at 0x13695544>

# now we add the Rectangle to the Axes
In [266]: ax.add_patch(rect)

# and notice that the ax.add_patch method has set the axes
# instance
In [267]: print rect.get_axes()
Axes(0.125,0.1;0.775x0.8)

# and the transformation has been set too
In [268]: print rect.get_transform()
<Affine object at 0x15009ca4>

# the default axes transformation is ax.transData
In [269]: print ax.transData
<Affine object at 0x15009ca4>

# notice that the xlimits of the Axes have not been changed
In [270]: print ax.get_xlim()
(0.0, 1.0)

# but the data limits have been updated to encompass the rectang
le
In [271]: print ax.dataLim.bounds
(1.0, 1.0, 5.0, 12.0)

# we can manually invoke the auto-scaling machinery
In [272]: ax.autoscale_view()

# and now the xlim are updated to encompass the rectangle
In [273]: print ax.get_xlim()
(1.0, 6.0)

# we have to manually force a figure draw
In [274]: ax.figure.canvas.draw()
```

有非常多的 `Axes` 辅助方法用于创建基本艺术家并将它们添加到他们各自的容器中。 下表总结了他们的一部分，他们创造的 `Artist` 的种类，以及他们在哪里存储它们。

辅助方法	艺术家	容器
`ax.annotate` - 文本标注	`Annotate`	`ax.texts`
`ax.bar` - 条形图	`Rectangle`	`ax.patches`
`ax.errorbar` - 误差条形图	`Line2D` 和 `Rectangle`	`ax.lines` 和 `ax.patches`
`ax.fill` - 共享区域	`Polygon`	`ax.patches`
`ax.hist` - 直方图	`Rectangle`	`ax.patches`
`ax.imshow` - 图像数据	`AxesImage`	`ax.images`
`ax.legend` - 轴域图例	`Legend`	`ax.legends`
`ax.plot` - xy 绘图	`Line2D`	`ax.lines`
`ax.scatter` - 散点图	`PolygonCollection`	`ax.collections`
`ax.text` - 文本	`Text`	`ax.texts`

除了所有这些艺术家， `Axes` 包含两个重要的艺术家容器： `XAxis` 和 `YAxis` ，它们处理刻度和标签的绘制。 它们被存储为实例变量 `xaxis` 和 `yaxis` 。 `XAxis` 和 `YAxis` 容器将在下面详细介绍，但请注意， `Axes` 包含许多辅助方法，它们会将调用转发给 `Axis` 实例，因此你通常不需要直接使用它们，除非你愿意。 例如，你可以使用 `Axes` 辅助程序方法设置 `XAxis` 刻度标签的字体大小：

```
for label in ax.get_xticklabels():
    label.set_color('orange')
```

下面是轴域所包含的艺术家的总结

轴域属性	描述
`artists`	`Artist` 实例的列表
`patch`	用于轴域背景的 `Rectangle` 实例
`collections`	`Collection` 实例的列表
`images`	`AxesImage` 的列表
`legends`	`Legend` 实例的列表
`lines`	`Line2D` 实例的列表
`patches`	`Patch` 实例的列表
`texts`	`Text` 实例的列表
`xaxis`	`matplotlib.axis.XAxis` 实例
`yaxis`	`matplotlib.axis.YAxis` 实例

轴容器

`matplotlib.axis.Axis` 实例处理刻度线，网格线，刻度标签和轴标签的绘制。你可以分别为y轴配置左和右刻度，为x轴分别配置上和下刻度。 `Axis` 还存储在自动缩放，平移和缩放中使用的数据和视图间隔，以及 `Locator` 和 `Formatter` 实例，它们控制刻度位置以及它们表示为字符串的方式。

每个 `Axis` 对象都包含一个 `label` 属性（这是 pylab 在调用 `xlabel()` 和 `ylabel()` 时修改的东西）以及主和次刻度的列表。刻度是 `XTick` 和 `YTick` 实例，它包含渲染刻度和刻度标签的实际线条和文本基本类型。因为刻度是按需动态创建的（例如，当平移和缩放时），你应该通过访问器方法 `get_major_ticks()` 和 `get_minor_ticks()` 访问主和次刻度的列表。虽然刻度包含所有下面要提及的基本类型， `Axis` 方法包含访问器方法来返回刻度线，刻度标签，刻度位置等：

```
In [285]: axis = ax.xaxis

In [286]: axis.get_ticklocs()
Out[286]: array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.
])

In [287]: axis.get_ticklabels()
Out[287]: <a list of 10 Text major ticklabel objects>

# note there are twice as many ticklines as labels because by
#  default there are tick lines at the top and bottom but only t
ick
#  labels below the xaxis; this can be customized
In [288]: axis.get_ticklines()
Out[288]: <a list of 20 Line2D ticklines objects>

# by default you get the major ticks back
In [291]: axis.get_ticklines()
Out[291]: <a list of 20 Line2D ticklines objects>

# but you can also ask for the minor ticks
In [292]: axis.get_ticklines(minor=True)
Out[292]: <a list of 0 Line2D ticklines objects>
```

下面是 `Axis` 的一些有用的访问器方法的总结（它们拥有相应的 `setter`，如 `set_major_formatter`）。

访问器方法	描述	
get_scale	轴的比例，例如 `'log'` 或 `'linear'`	
get_view_interval	轴视图范围的内部实例	
get_data_interval	轴数据范围的内部实例	
get_gridlines	轴的网格线列表	
get_label	轴标签 - `Text` 实例	
get_ticklabels	`Text` 实例的列表 - 关键字`minor=True	False`
get_ticklines	`Line2D` 实例的列表 - 关键字`minor=True	False`
get_ticklocs	`Tick` 位置的列表 - 关键字`minor=True	False`
get_major_locator	用于主刻度的 `matplotlib.ticker.Locator` 实例	
get_major_formatter	用于主刻度的 `matplotlib.ticker.Formatter` 实例	
get_minor_locator	用于次刻度的 `matplotlib.ticker.Locator` 实例	
get_minor_formatter	用于次刻度的 `matplotlib.ticker.Formatter` 实例	
get_major_ticks	用于主刻度的 `Tick` 实例列表	
get_minor_ticks	用于次刻度的 `Tick` 实例列表	
grid	为主或次刻度打开或关闭网格	

这里是个例子，出于美观不太推荐，它自定义了轴域和刻度属性。

```
import numpy as np
import matplotlib.pyplot as plt

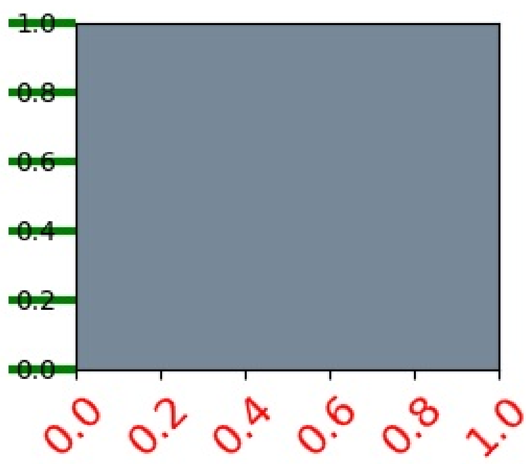
# plt.figure creates a matplotlib.figure.Figure instance
fig = plt.figure()
rect = fig.patch # a rectangle instance
rect.set_facecolor('lightgoldenrodyellow')

ax1 = fig.add_axes([0.1, 0.3, 0.4, 0.4])
rect = ax1.patch
rect.set_facecolor('lightslategray')


for label in ax1.xaxis.get_ticklabels():
    # label is a Text instance
    label.set_color('red')
    label.set_rotation(45)
    label.set_fontsize(16)

for line in ax1.yaxis.get_ticklines():
    # line is a Line2D instance
    line.set_color('green')
    line.set_markersize(25)
    line.set_markeredgewidth(3)

plt.show()
```



刻度容器

`matplotlib.axis.Tick` 是我们从 `Figure` 到 `Axes` 再到 `Axis` 再到 `Tick` 的最终的容器对象。 `Tick` 包含刻度和网格线的实例，以及上侧和下侧刻度的标签实例。 每个都可以直接作为 `Tick` 的属性访问。此外，也有用于确定上标签和刻度是否对应 `x` 轴，以及右标签和刻度是否对应 `y` 轴的布尔变量。

刻度属性	描述
`tick1line`	`Line2D` 实例
`tick2line`	`Line2D` 实例
`gridline`	`Line2D` 实例
`label1`	`Text` 实例
`label2`	`Text` 实例
`gridOn`	确定是否绘制刻度线的布尔值
`tick1On`	确定是否绘制主刻度线的布尔值
`tick2On`	确定是否绘制次刻度线的布尔值
`label1On`	确定是否绘制主刻度标签的布尔值
`label2On`	确定是否绘制次刻度标签的布尔值

这里是个例子，使用美元符号设置右侧刻度，并在 `y` 轴右侧将它们设成绿色。

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

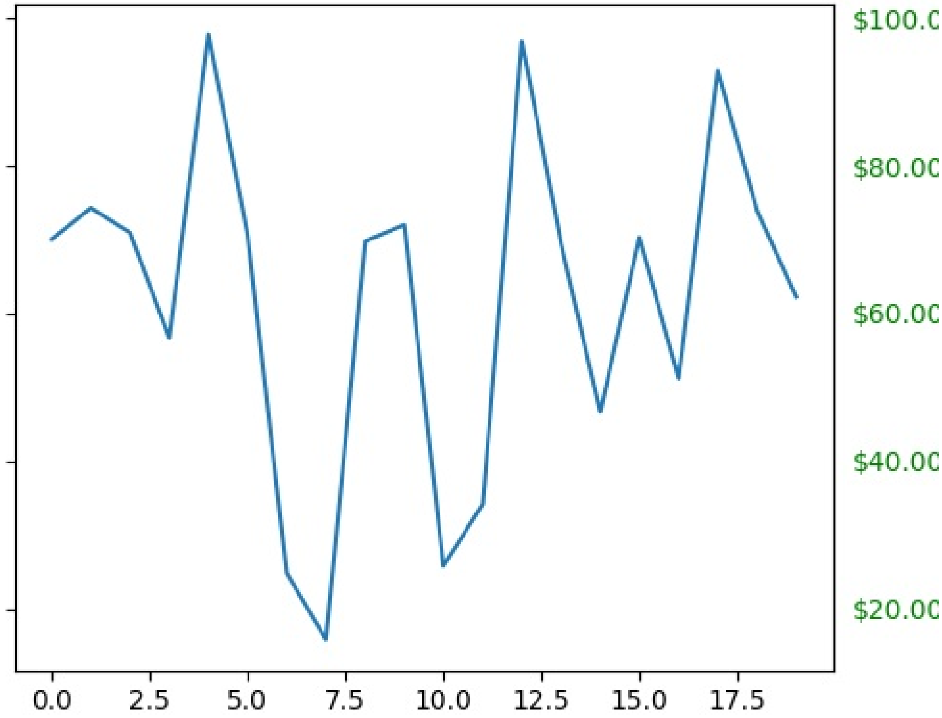
# Fixing random state for reproducibility
np.random.seed(19680801)

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(100*np.random.rand(20))

formatter = ticker.FormatStrFormatter('$%1.2f')
ax.yaxis.set_major_formatter(formatter)

for tick in ax.yaxis.get_major_ticks():
    tick.label1On = False
    tick.label2On = True
    tick.label2.set_color('green')

plt.show()
```


$100.0
$80.00
$60.00
$40.00
$20.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

图例指南

> 原文：Legend guide
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

此图例指南是 `legend()` 中可用文档的扩展 - 请在继续阅读本指南之前确保你熟悉该文档（见篇尾）的内容。

本指南使用一些常见术语，为了清楚起见，这些术语在此处进行说明：

图例条目

图例由一个或多个图例条目组成。 一个条目由一个键和一个标签组成。

图例键

每个图例标签左侧的彩色/图案标记。

图例标签

描述由键表示的句柄的文本。

图例句柄

用于在图例中生成适当条目的原始对象。

控制图例条目

不带参数调用 `legend()` 会自动获取图例句柄及其相关标签。 此函数等同于：

```
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles, labels)
```

`get_legend_handles_labels()` 函数返回轴域上存在的句柄/艺术家的列表，这些句柄/艺术家可以用于为结果图例生成条目 - 但值得注意的是，并非所有艺术家都可以添加到图例中， 这种情况下会创建『代理』（请参阅特地为添加到图例创建艺术家（也称为代理艺术家），来了解更多详细信息）。

为了完全控制要添加到图例的内容，通常将适当的句柄直接传递给 `legend()` ：

```
line_up, = plt.plot([1,2,3], label='Line 2')
line_down, = plt.plot([3,2,1], label='Line 1')
plt.legend(handles=[line_up, line_down])
```

在某些情况下，不可能设置句柄的标签，因此可以将标签列表传递给 `legend()` ：

```
line_up, = plt.plot([1,2,3], label='Line 2')
line_down, = plt.plot([3,2,1], label='Line 1')
plt.legend([line_up, line_down], ['Line Up', 'Line Down'])
```

特地为添加到图例创建艺术家（也称为代理艺术家）

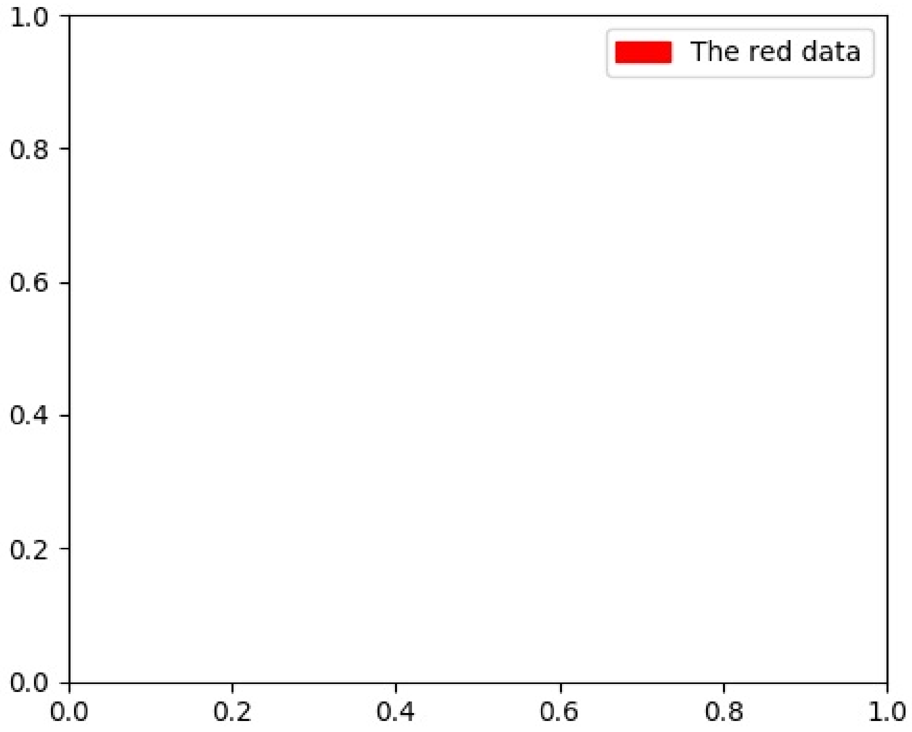
并非所有的句柄都可以自动转换为图例条目，因此通常需要创建一个可转换的艺术家。 图例句柄不必存在于被用到的图像或轴域上。

假设我们想创建一个图例，其中有一些数据表示为红色：

```
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt

red_patch = mpatches.Patch(color='red', label='The red data')
plt.legend(handles=[red_patch])

plt.show()
```



除了创建一个色块之外，有许多受支持的图例句柄，我们可以创建一个带有标记的线条：

```
import matplotlib.lines as mlines
import matplotlib.pyplot as plt

blue_line = mlines.Line2D([], [], color='blue', marker='*',
                          markersize=15, label='Blue stars')
plt.legend(handles=[blue_line])

plt.show()
```

图例位置

图例的位置可以通过关键字参数 `loc` 指定。详细信息请参阅 `legend()` 的文档。

`bbox_to_anchor` 关键字可让用户手动控制图例布局。例如，如果你希望轴域图例位于图像的右上角而不是轴域的边角，则只需指定角的位置以及该位置的坐标系：

```
plt.legend(bbox_to_anchor=(1, 1),
           bbox_transform=plt.gcf().transFigure)
```

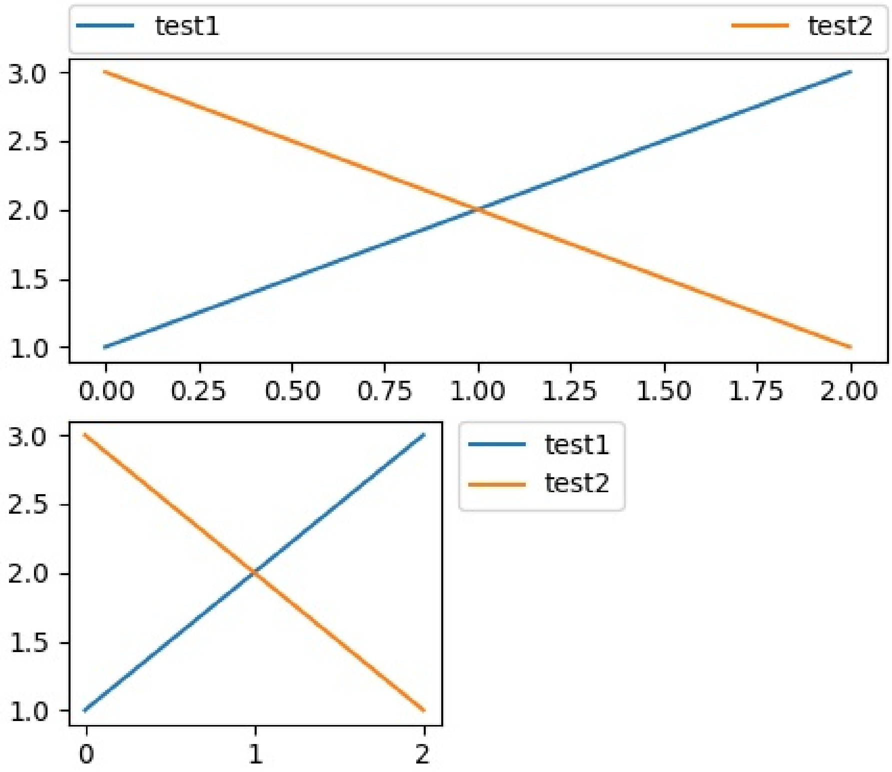
自定义图例位置的更多示例：

```
import matplotlib.pyplot as plt


plt.subplot(211)
plt.plot([1,2,3], label="test1")
plt.plot([3,2,1], label="test2")
# 将图例放到这个子图上方，
# 扩展自身来完全利用提供的边界框。
plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
           ncol=2, mode="expand", borderaxespad=0.)

plt.subplot(223)
plt.plot([1,2,3], label="test1")
plt.plot([3,2,1], label="test2")
# 将图例放到这个小型子图的右侧
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.show()
```



相同轴域内的多个图例

有时，在多个图例之间分割图例条目会更加清晰。 虽然直觉上的做法可能是多次调用 `legend()` 函数，但你会发现轴域上只存在一个图例。 这样做是为了可以重复调用 `legend()` ，将图例更新为轴域上的最新句柄，因此要保留旧的图例实例，我们必须将它们手动添加到轴域中：

```
import matplotlib.pyplot as plt

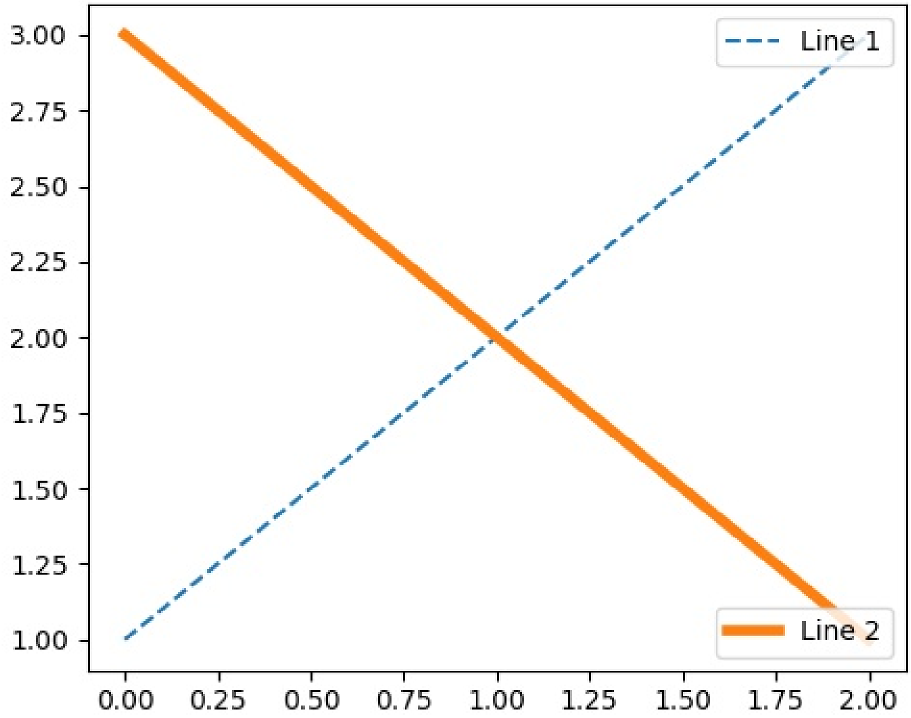
line1, = plt.plot([1,2,3], label="Line 1", linestyle='--')
line2, = plt.plot([3,2,1], label="Line 2", linewidth=4)

# 为第一个线条创建图例
first_legend = plt.legend(handles=[line1], loc=1)

# 手动将图例添加到当前轴域
ax = plt.gca().add_artist(first_legend)

# 为第二个线条创建另一个图例
plt.legend(handles=[line2], loc=4)

plt.show()
```



图例处理器

为了创建图例条目，将句柄作为参数提供给适当的 `HandlerBase` 子类。 处理器子类的选择由以下规则确定：

- 使用 `handler_map` 关键字中的值更新 `get_legend_handler_map()` 。
- 检查句柄是否在新创建的 `handler_map` 中。
- 检查句柄的类型是否在新创建的 `handler_map` 中。
- 检查句柄的 `mro` 中的任何类型是否在新创建的 `handler_map` 中。

处于完整性，这个逻辑大多在 `get_legend_handler()` 中实现。

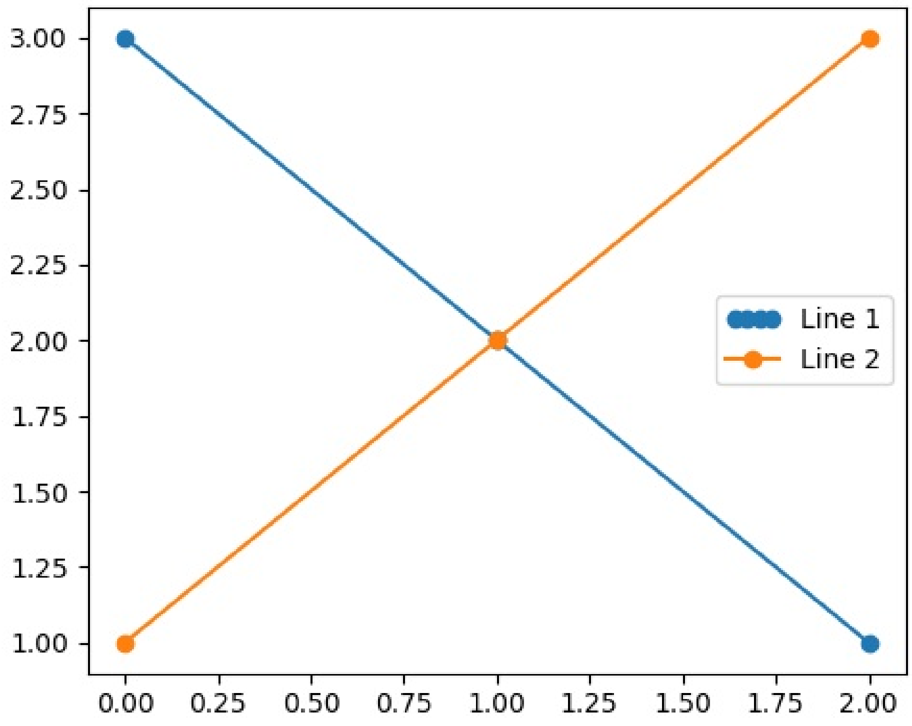
所有这些灵活性意味着我们可以使用一些必要的钩子，为我们自己的图例键类型实现自定义处理器。

使用自定义处理器的最简单的例子是，实例化一个现有的 `HandlerBase` 子类。 为了简单起见，让我们选择 `matplotlib.legend_handler.HandlerLine2D` ，它接受 `numpoints` 参数（出于便利，注意 `numpoints` 是 `legend()` 函数上的一个关键字）。 然后我们可以将实例的字典作为关键字 `handler_map` 传给 `legend` 。

```
import matplotlib.pyplot as plt
from matplotlib.legend_handler import HandlerLine2D

line1, = plt.plot([3,2,1], marker='o', label='Line 1')
line2, = plt.plot([1,2,3], marker='o', label='Line 2')

plt.legend(handler_map={line1: HandlerLine2D(numpoints=4)})
```



如你所见， `Line 1` 现在有 4 个标记点， `Line 2` 有两个（默认值）。 尝试上面的代码，只需将字典的键从 `line1` 更改为type(line) `。 注意现在两个` Line2D`实例都拥有了 4 个标记。

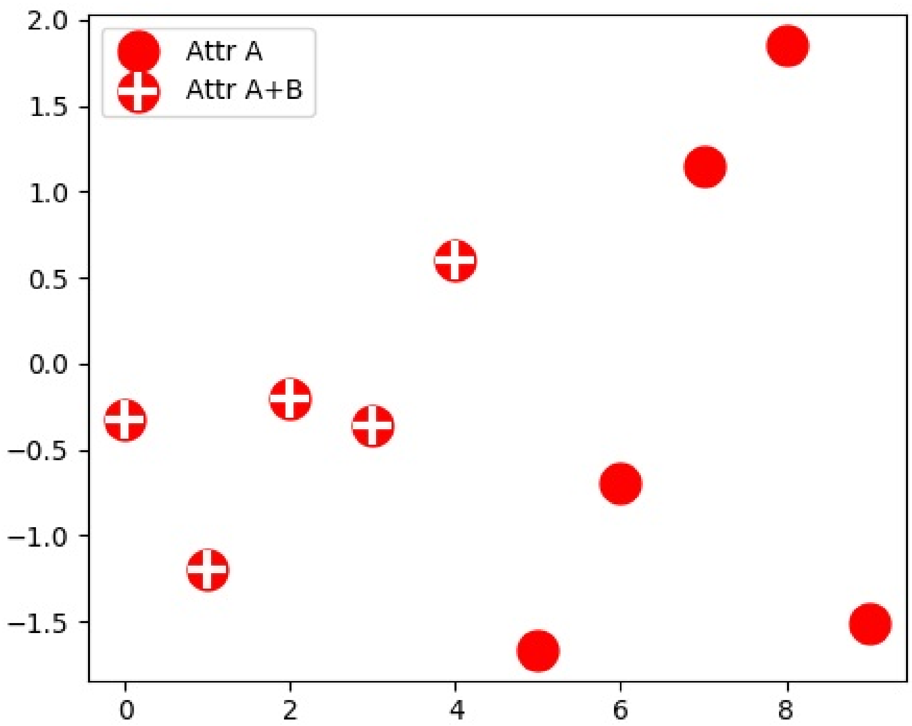
除了用于复杂的绘图类型的处理器，如误差条，茎叶图和直方图，默认的 `handler_map` 有一个特殊的元组处理器（ `HandlerTuple` ），它简单地在顶部一一绘制给定元组中每个项目的句柄。 以下示例演示如何将两个图例的键相互叠加：

```
import matplotlib.pyplot as plt
from numpy.random import randn

z = randn(10)

red_dot, = plt.plot(z, "ro", markersize=15)
# 将白色十字放置在一些数据上
white_cross, = plt.plot(z[:5], "w+", markeredgewidth=3, markersi
ze=15)

plt.legend([red_dot, (red_dot, white_cross)], ["Attr A", "Attr A
+B"])
```



实现自定义图例处理器

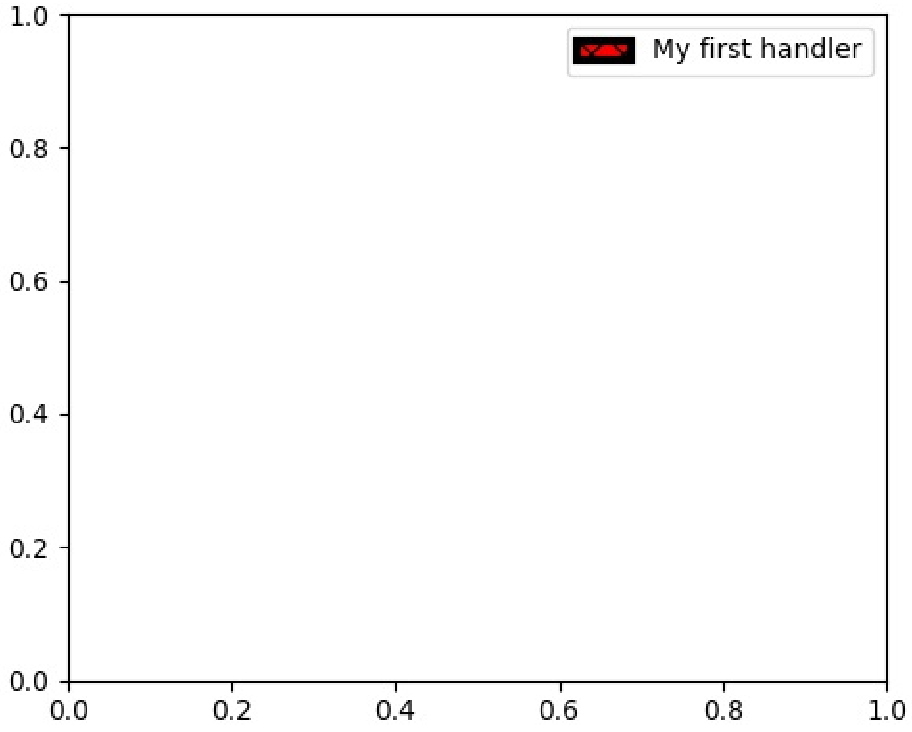
可以实现自定义处理器，将任何句柄转换为图例的键（句柄不必要是 `matplotlib` artist）。处理器必须实现 `legend_artist` 方法，该方法为要使用的图例返回单个艺术家。有关 `legend_artist` 的详细信息，请参阅 `legend_artist()` 。

```
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

class AnyObject(object):
    pass

class AnyObjectHandler(object):
    def legend_artist(self, legend, orig_handle, fontsize, handl
ebox):
        x0, y0 = handlebox.xdescent, handlebox.ydescent
        width, height = handlebox.width, handlebox.height
        patch = mpatches.Rectangle([x0, y0], width, height, face
color='red',
                                   edgecolor='black', hatch='xx'
, lw=3,
                                   transform=handlebox.get_trans
form())
        handlebox.add_artist(patch)
        return patch

plt.legend([AnyObject()], ['My first handler'],
           handler_map={AnyObject: AnyObjectHandler()})
```



或者，如果我们想要接受全局的 `AnyObject` 实例，而不想一直手动设置 `handler_map` 关键字，我们可以注册新的处理器：

```
from matplotlib.legend import Legend
Legend.update_default_handler_map({AnyObject: AnyObjectHandler()
})
```

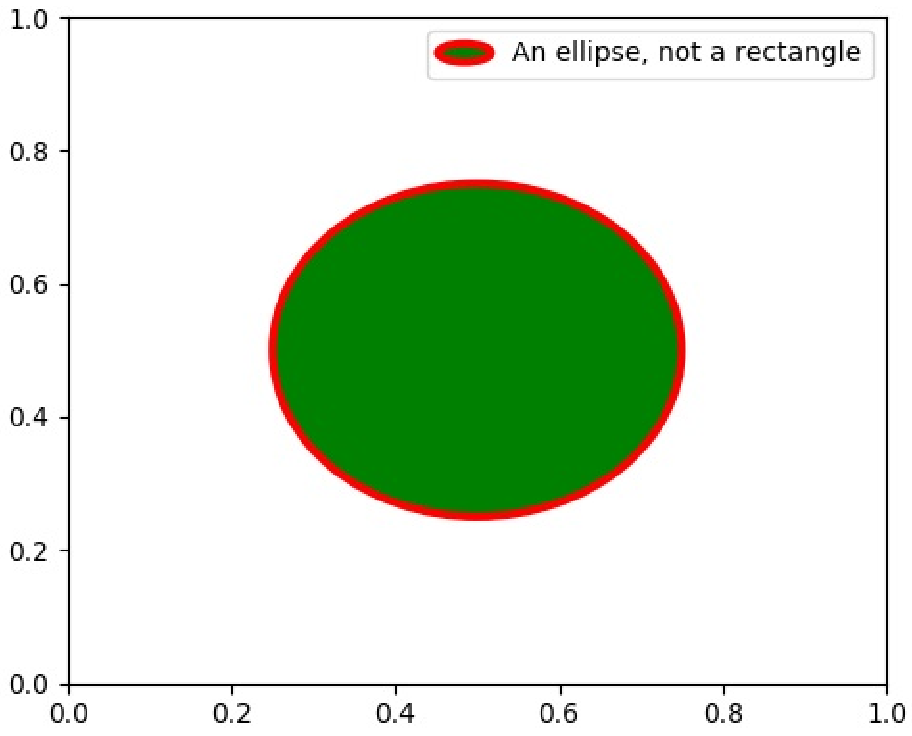
虽然这里的功能十分清楚，请记住，有很多已实现的处理器，你想实现的目标可能易于使用现有的类实现。 例如，要生成椭圆的图例键，而不是矩形键：

```
from matplotlib.legend_handler import HandlerPatch
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches


class HandlerEllipse(HandlerPatch):
    def create_artists(self, legend, orig_handle,
                       xdescent, ydescent, width, height, fontsi
ze, trans):
        center = 0.5 * width - 0.5 * xdescent, 0.5 * height - 0.5
 * ydescent
        p = mpatches.Ellipse(xy=center, width=width + xdescent,
                             height=height + ydescent)
        self.update_prop(p, orig_handle, legend)
        p.set_transform(trans)
        return [p]


c = mpatches.Circle((0.5, 0.5), 0.25, facecolor="green",
                    edgecolor="red", linewidth=3)
plt.gca().add_patch(c)

plt.legend([c], ["An ellipse, not a rectangle"],
           handler_map={mpatches.Circle: HandlerEllipse()})
```



使用图例的现有示例

这里是一个不太详尽的示例列表，涉及以各种方式使用的图例：

- `lines_bars_and_markers` 示例代码: `scatter_with_legend.py`
- API 示例代码: `legend_demo.py`
- `pylab_examples` 示例代码: `contourf_hatching.py`
- `pylab_examples` 示例代码: `figlegend_demo.py`
- `pylab_examples` 示例代码: `finance_work2.py`
- `pylab_examples` 示例代码: `scatter_symbol.py`

`matplotlib.pyplot.legend(*args, **kwargs)` 文档

在轴域上放置一个图例。

为了为轴域上已经存在的线条（例如通过绘图）制作图例，只需使用字符串的可迭代对象（每个图例条目对应一个字符串）调用此函数。 例如：

```
ax.plot([1, 2, 3])
ax.legend(['A simple line'])
```

但是，为了使『标签』和图例元素实例保持一致，最好在艺术家创建时指定标签，或者通过调用艺术家的 `set_label()` 方法：

```
line, = ax.plot([1, 2, 3], label='Inline label')
# 通过调用该方法覆写标签
line.set_label('Label via method')
ax.legend()
```

通过定义以下划线开头的标签，可以从图例元素自动选择中排除特定线条。 这对于所有艺术家都是默认的，因此不带任何参数调用 `legend()` ，并且没有手动设置标签会导致没有绘制图例。

为了完全控制哪些艺术家拥有图例条目，可以传递拥有图例的艺术家的可迭代对象，然后是相应图例标签的可迭代对象：

```
legend((line1, line2, line3), ('label1', 'label2', 'label3'))
```

参数

`loc` ：整数、字符串或者浮点偶对，默认为 `'upper right'` 。

图例的位置。 可能的代码是：

位置字符串	位置代码
`'best'`	`0`
`'upper right'`	`1`
`'upper left'`	`2`
`'lower left'`	`3`
`'lower right'`	`4`
`'right'`	`5`
`'center left'`	`6`
`'center right'`	`7`
`'lower center'`	`8`
`'upper center'`	`9`
`'center'`	`10`

或者，可以是一个二元组，提供图例的距离左下角的 `x, y` 坐标（在这种情况下， `bbox_to_anchor` 将被忽略）。

`bbox_to_anchor` ： `matplotlib.transforms.BboxBase` 示例或者浮点元组。

在 `bbox_transform` 坐标（默认轴域坐标）中为图例指定任意位置。

例如，要将图例的右上角放在轴域中心，可以使用以下关键字：

```
loc='upper right', bbox_to_anchor=(0.5, 0.5)
```

`ncol` ：整数。

图例的列数，默认为 1。

`prop` ： `None` 、 `matplotlib.font_manager.FontProperties` 或者字典。

图例的字体属性，如果为 `None` （默认），会使用当前的 `matplotlib.rcParams` 。

`fontsize` ：整数、浮点或者 `{‘xx-small’, ‘x-small’, ‘small’, ‘medium’, ‘large’, ‘x-large’, ‘xx`。

控制图例的字体大小。 如果值为数字，则大小将为绝对字体大小（以磅为单位）。 字符串值相对于当前默认字体大小。 此参数仅在未指定 `prop` 的情况下使用。

`numpoints` ： `None` 或者整数。

为线条/ `matplotlib.lines.Line2D` 创建图例条目时，图例中的标记点数。 默认值为 `None` ，它将从 `legend.numpoints` `rcParam` 中获取值。

`scatterpoints` ： `None` 或者整数。

为散点图/ `matplotlib.collections.PathCollection` 创建图例条目时，图例中的标记点数。 默认值为 `None` ，它将从 `legend.scatterpoints` `rcParam` 中获取值。

`scatteryoffsets` ：浮点的可迭代对象。

为散点图图例条目创建的标记的垂直偏移量（相对于字体大小）。 0.0 是在图例文本的底部，1.0 是在顶部。 为了将所有标记绘制在相同的高度，请设置为 `[0.5]` 。 默认值为 `[0.375,0.5,0.3125]` 。

`markerscale` ： `None` 、整数或者浮点。

图例标记对于原始绘制的标记的相对大小。 默认值为 `None` ，它将从 `legend.markerscale` `rcParam` 中获取值。

`markerfirst` : `[ True | False ]`

如果为 `True` ，则图例标记位于图例标签的左侧，如果为 `False` ，图例标记位于图例标签的右侧。

`frameon` ： `None` 或布尔值

控制是否应在图例周围绘制框架。默认值为 `None` ，它将从 `legend.frameon` `rcParam` 中获取值。

`fancybox` ： `None` 或布尔值

控制是否应在构成图例背景的 `FancyBboxPatch` 周围启用圆边。默认值为 `None` ，它将从 `legend.fancybox` `rcParam` 中获取值。

`shadow` ： `None` 或布尔值

控制是否在图例后面画一个阴影。默认值为 `None` ，它将从 `legend.shadow` `rcParam` 中获取值。

`framealpha` ： `None` 或浮点

控制图例框架的 Alpha 透明度。默认值为 `None` ，它将从 `legend.framealpha` `rcParam` 中获取值。

`mode` ： `{"expand", None}`

如果 `mode` 设置为 `"expand"` ，图例将水平扩展来填充轴域区域（如果定义图例的大小，则为 `bbox_to_anchor` ）。

`bbox_transform` ： `None` 或者 `matplotlib.transforms.Transform`

边界框的变换（ `bbox_to_anchor` ）。对于 `None` 值（默认），将使用 `Axes` 的 `transAxes` 变换。

`title` ：字符串或者 `None`

图例的标题，默认没有标题（ `None` ）。

`borderpad` ：浮点或 `None`

图例边框的内边距。以字体大小为单位度量。默认值为 `None` ，它将从 `legend.borderpad` `rcParam` 中获取值。

`labelspacing` ：浮点或 `None`

图例条目之间的垂直间距。以字体大小为单位度量。默认值为 `None` ，它将从 `legend.labelspacing` `rcParam` 中获取值。

`handlelength` ：浮点或 `None`

图例句柄的长度。以字体大小为单位度量。默认值为 `None` ，它将从 `legend.handlelength` `rcParam` 取值。

`handletextpad` ：浮点或 `None`

图例句柄和文本之间的间距。以字体大小为单位度量。默认值为 `None` ，它将从 `legend.handletextpad` `rcParam` 中获取值。

`borderaxespad` ：浮点或 `None`

轴和图例边框之间的间距。以字体大小为单位度量。默认值为 `None`，它将从 `legend.borderaxespad` `rcParam` 中获取值。

`columnspacing`：浮点或 `None`

列间距。以字体大小为单位度量。默认值为 `None`，它将从 `legend.columnspacing` `rcParam` 中获取值。

`handler_map`：字典或 `None`

自定义字典，用于将实例或类型映射到图例处理器。这个 `handler_map` 会更新在 `matplotlib.legend.Legend.get_legend_handler_map()` 中获得的默认处理器字典。

变换教程

> 原文：Transformations Tutorial
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

像任何图形包一样，matplotlib 建立在变换框架之上，以便在坐标系，用户数据坐标系，轴域坐标系，图形坐标系和显示坐标系之间轻易变换。 在 95 ％的绘图中，你不需要考虑这一点，因为它发生在背后，但随着你接近自定义图形生成的极限，它有助于理解这些对象，以便可以重用 matplotlib 提供给你的现有变换，或者创建自己的变换（见 `matplotlib.transforms` ）。 下表总结了现有的坐标系，你应该在该坐标系中使用的变换对象，以及该系统的描述。 在『变换对象』一列中， `ax` 是 `Axes` 实例， `fig` 是一个图形实例。

坐标系	变换对象	描述
数据	`ax.transData`	用户数据坐标系，由 `xlim` 和 `ylim` 控制
轴域	`ax.transAxes`	轴域坐标系； `(0,0)` 是轴域左下角， `(1,1)` 是轴 上角
图形	`fig.transFigure`	图形坐标系； `(0,0)` 是图形左下角， `(1,1)` 是图 上角
显示	`None`	这是显示器的像素坐标系; `(0,0)` 是显示器的左下角， `(width, height)` 是显示器的右上角，以像素单位。 或者，可以使用恒等变换（ `matplotlib.transforms.IdentityTransform` 来代替 `None` 。

上表中的所有变换对象都接受以其坐标系为单位的输入，并将输入变换到显示坐标系。 这就是为什么显示坐标系没有『变换对象』的原因 - 它已经以显示坐标为单位了。 变换也知道如何反转自身，从显示返回自身的坐标系。 这在处理来自用户界面的事件（通常发生在显示空间中），并且你想知道数据坐标系中鼠标点击或按键按下的位置时特别有用。

数据坐标

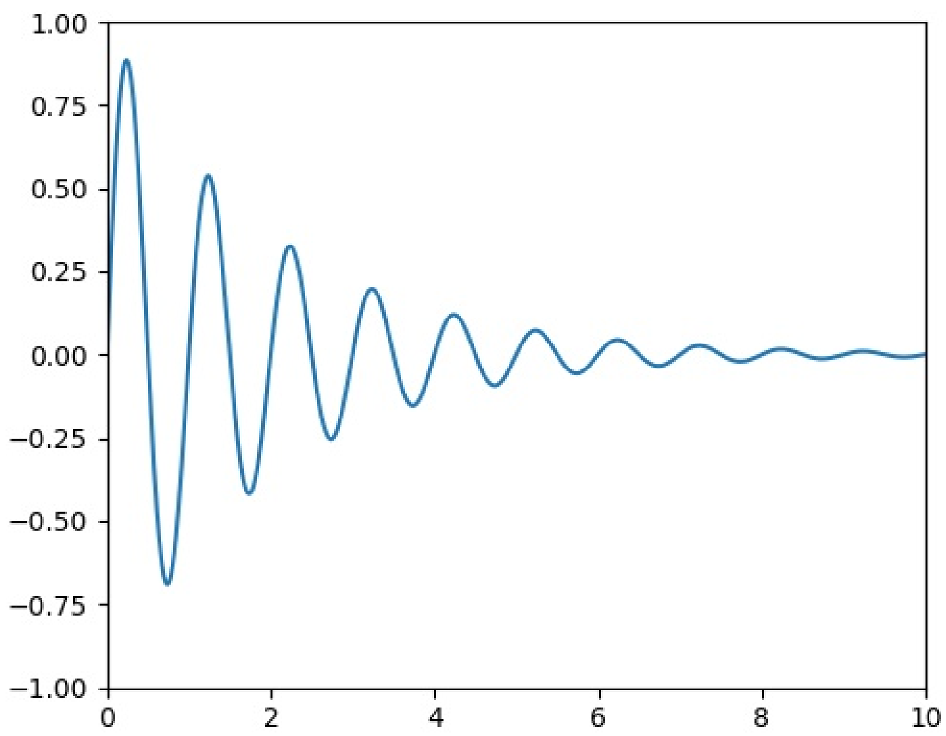
让我们从最常用的坐标，数据坐标系开始。 每当向轴域添加数据时，matplotlib 会更新数据对象， `set_xlim()` 和 `set_ylim()` 方法最常用于更新。 例如，在下图中，数据的范围在 `x` 轴上为从 0 到 10，在 `y` 轴上为从 -1 到 1。

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 10, 0.005)
y = np.exp(-x/2.) * np.sin(2*np.pi*x)

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(x, y)
ax.set_xlim(0, 10)
ax.set_ylim(-1, 1)

plt.show()
```



你可以使用 `ax.transData` 实例将数据变换为显示坐标系，无论是单个点或是一系列点，如下所示：

```
In [14]: type(ax.transData)
Out[14]: <class 'matplotlib.transforms.CompositeGenericTransform
'>

In [15]: ax.transData.transform((5, 0))
Out[15]: array([ 335.175,  247.   ])

In [16]: ax.transData.transform([(5, 0), (1,2)])
Out[16]:
array([[ 335.175,  247.   ],
       [ 132.435,  642.2  ]])
```

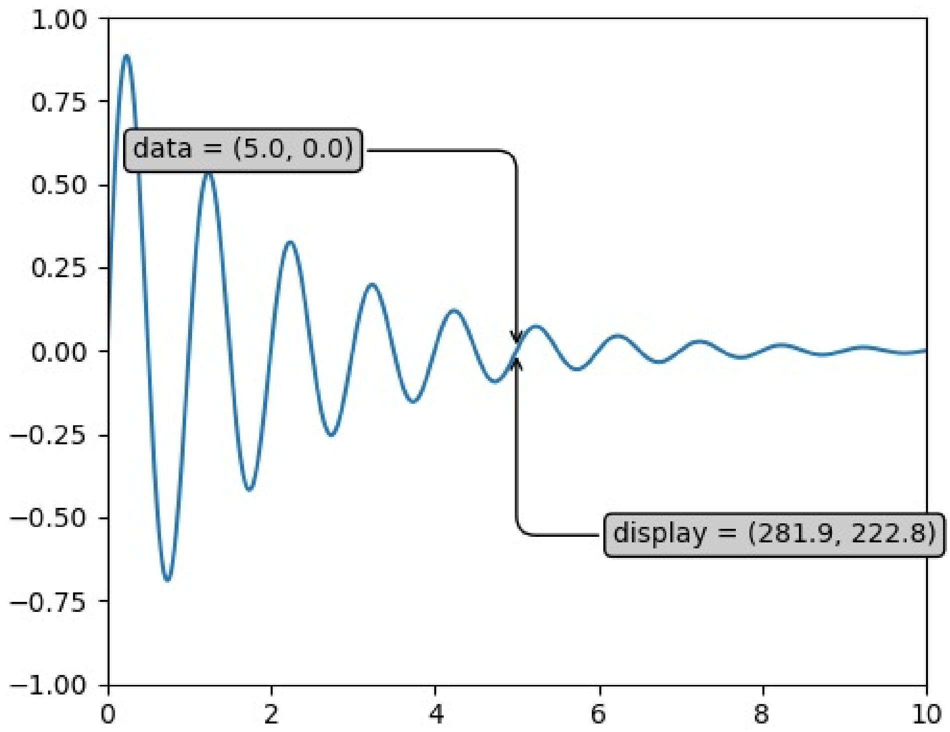
你可以使用 `inverted()` 方法创建一个变换，从显示坐标变换为数据坐标：

```
In [41]: inv = ax.transData.inverted()

In [42]: type(inv)
Out[42]: <class 'matplotlib.transforms.CompositeGenericTransform
'>

In [43]: inv.transform((335.175,  247.))
Out[43]: array([ 5.,  0.])
```

如果你一直关注本教程，如果你的窗口大小或 dpi 设置不同，显示坐标的确切值可能会有所不同。 同样，在下面的图形中，在 ipython 会话中，由显示标记的点可能并不相同，因为文档图形大小默认值是不同的。



> 注意
>
> 如果在 GUI 后端中运行上述示例中的源代码，你还可能发现数据和显示标注的两个箭头不会指向完全相同的点。这是因为显示点是在显示图形之前计算的，并且 GUI 后端可以在创建图形时稍微调整图形大小。如果你自己调整图的大小，效果更明显。这是你很少想要处理显示空间的一个很好的原因，但是你可以连接到 `'on_draw'` 事件来更新图上的图坐标；请参阅事件处理和选择。

当你更改轴的 `x` 或 `y` 的范围时，将更新数据范围，以便变换生成新的显示点。注意，当我们只是改变 `ylim`，只有 `y` 显示坐标改变，当我们改变 `xlim` 也同理。我们在谈论 Bbox 时会深入。

```
In [54]: ax.transData.transform((5, 0))
Out[54]: array([ 335.175,  247.    ])

In [55]: ax.set_ylim(-1,2)
Out[55]: (-1, 2)

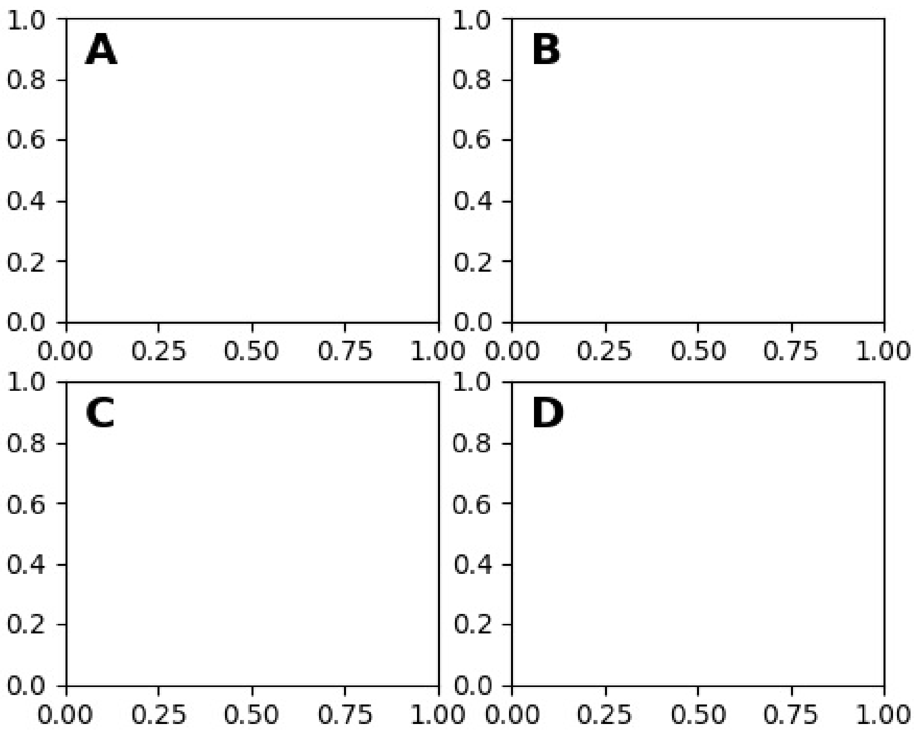
In [56]: ax.transData.transform((5, 0))
Out[56]: array([ 335.175     ,  181.13333333])

In [57]: ax.set_xlim(10,20)
Out[57]: (10, 20)

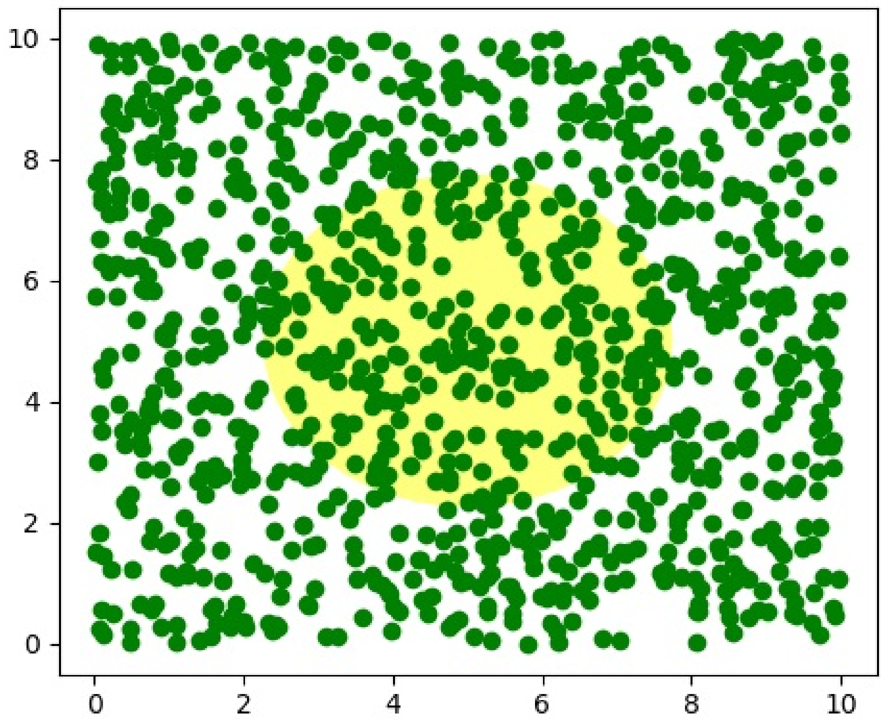
In [58]: ax.transData.transform((5, 0))
Out[58]: array([-171.675     ,  181.13333333])
```

轴域坐标

在数据坐标系之后，轴域可能是第二有用的坐标系。 这里，点 `(0,0)` 是轴域或子图的左下角， `(0.5,0.5)` 是中心， `(1.0,1.0)` 是右上角。 你还可以引用范围之外的点，因此 `(-0.1,1.1)` 位于轴的左上方。 此坐标系在将文本放置在轴中时非常有用，因为你通常需要在固定的位置（例如，轴域窗格的左上角）放置文本气泡，并且在平移或缩放时保持该位置固定。 这里是一个简单的例子，创建四个面板，并将他们标记为 `'A'` ， `'B'` ， `'C'` ， `'D'` ，你经常在期刊上看到它们。



你也可以在轴坐标系中创建线条或者补丁，但是以我的经验，这比使用 `ax.transAxes` 放置文本更不实用。 尽管如此，这里是一个愚蠢的例子，它在数据空间中绘制了一些随机点，并且覆盖在一个半透明的圆上面，这个圆以轴域的中心为圆心，半径为轴域的四分之一。 - 如果你的轴域不保留高宽比（见 `set_aspect()` ），它将看起来像一个椭圆。 使用平移/缩放工具移动，或手动更改数据的 `xlim` 和 `ylim` ，你将看到数据移动，但圆将保持固定，因为它不在数据坐标中，并且将始终保持在轴域的中心 。



混合变换

在数据与轴域坐标混合的混合坐标空间中绘制是非常实用的，例如创建一个水平跨度，突出 `y` 数据的一些区域但横跨 `x` 轴，而无论数据限制，平移或缩放级别等。实际上这些混合线条和跨度非常有用，我们已经内置了一些函数来使它们容易绘制（参见 `axhline()` ， `axvline()` ， `axhspan()` ， `axvspan()` ），但是为了教学目的，我们使用混合变换实现这里的水平跨度。这个技巧只适用于可分离的变换，就像你在正常的笛卡尔坐标系中看到的，但不能为不可分离的变换，如 `PolarTransform` （极坐标变换）。

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.transforms as transforms

fig = plt.figure()
ax = fig.add_subplot(111)

x = np.random.randn(1000)

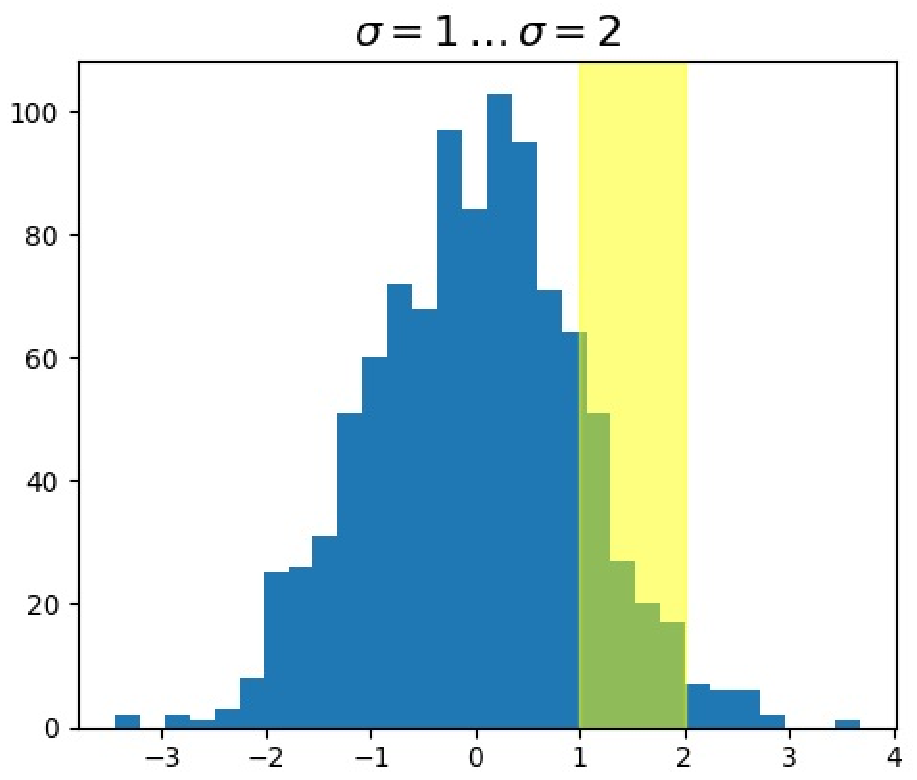
ax.hist(x, 30)
ax.set_title(r'$\sigma=1 \/ \dots \/ \sigma=2$', fontsize=16)

# the x coords of this transformation are data, and the
# y coord are axes
trans = transforms.blended_transform_factory(
    ax.transData, ax.transAxes)

# highlight the 1..2 stddev region with a span.
# We want x to be in data coordinates and y to
# span from 0..1 in axes coords
rect = patches.Rectangle((1,0), width=1, height=1,
                         transform=trans, color='yellow',
                         alpha=0.5)

ax.add_patch(rect)

plt.show()
```



> 注
>
> 混合变换非常有用，其中 `x` 为数据坐标而 `y` 为轴域坐标，我们拥有辅助方法来返回内部使用的版本 mpl ，用于绘制 `ticks` ， `ticklabels` 以及其他。方法是 `matplotlib.axes.Axes.get_xaxis_transform()` 和 `matplotlib.axes.Ax` 。因此，在上面的示例中， `blended_transform_factory()` 的调用可以替换为 `get_xaxis_transform` ：
>
> ```
> trans = ax.get_xaxis_transform()
> ```

使用偏移变换来创建阴影效果

变换的一个用法，是创建偏离另一变换的新变换，例如，放置一个对象，相对于另一对象有一些偏移。 通常，你希望物理尺寸上有一些移位，例如以点或英寸，而不是数据坐标为单位，以便移位效果在不同的缩放级别和 dpi 设置下保持不变。

偏移的一个用途是创建一个阴影效果，其中你绘制一个与第一个相同的对象，刚好在它的右边和下面，调整 `zorder` 来确保首先绘制阴影，然后绘制对象，阴影在它之上。 变换模块具有辅助变换 `ScaledTranslation` 。 它可以这样来实例化：

```
trans = ScaledTranslation(xt, yt, scale_trans)
```

其中 `xt` 和 `yt` 是变换的偏移，`scale_trans` 是变换，在应用偏移之前的变换期间缩放 `xt` 和 `yt` 。一个典型的用例是，将图形的 `fig.dpi_scale_trans` 变换用于 `scale_trans` 参数，来在实现最终的偏移之前，首先将以点为单位的 `xt` 和 `yt` 缩放到显示空间。DPI 和英寸偏移是常见的用例，我们拥有一个特殊的辅助函数，来在 `matplotlib.transforms.offset_copy()` 中创建它，它返回一个带有附加偏移的新变换。但在下面的示例中，我们将自己创建偏移变换。注意使用加法运算符：

```
offset = transforms.ScaledTranslation(dx, dy,
         fig.dpi_scale_trans)
shadow_transform = ax.transData + offset
```

这里显示了，可以使用加法运算符将变换链起来。该代码表示：首先应用数据变换 `ax.transData`，然后由 `dx` 和 `dy` 点翻译数据。在排版中，一个点是 1/72 英寸，通过以点为单位指定偏移，你的图形看起来是一样的，无论所保存的 dpi 分辨率。

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.transforms as transforms

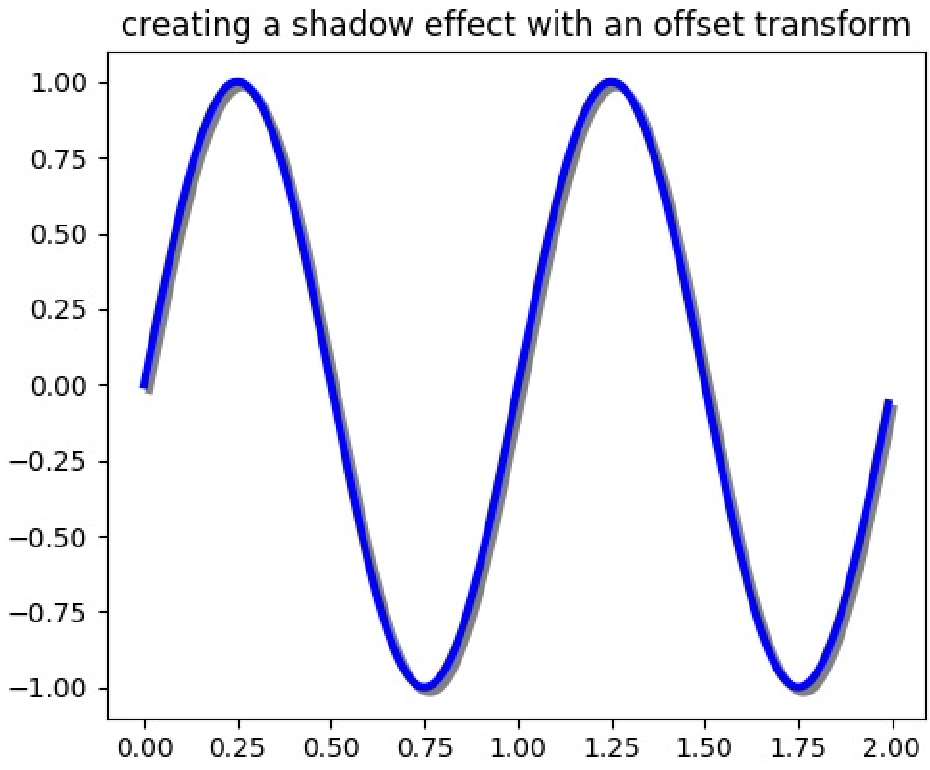
fig = plt.figure()
ax = fig.add_subplot(111)

# make a simple sine wave
x = np.arange(0., 2., 0.01)
y = np.sin(2*np.pi*x)
line, = ax.plot(x, y, lw=3, color='blue')

# shift the object over 2 points, and down 2 points
dx, dy = 2/72., -2/72.
offset = transforms.ScaledTranslation(dx, dy,
  fig.dpi_scale_trans)
shadow_transform = ax.transData + offset

# now plot the same data with our offset transform;
# use the zorder to make sure we are below the line
ax.plot(x, y, lw=3, color='gray',
  transform=shadow_transform,
  zorder=0.5*line.get_zorder())

ax.set_title('creating a shadow effect with an offset transform'
)
plt.show()
```



变换流水线

我们在本教程中一直使用的 `ax.transData` 变换是三种不同变换的组合，它们构成从数据到显示坐标的变换流水线。 Michael Droettboom 实现了变换框架，提供了一个干净的 API，它隔离了在极坐标和对数坐标图中发生的非线性投影和尺度，以及在平移和缩放时发生的线性仿射变换。 这里有一个效率问题，因为你可以平移和放大你的轴域，它会影响仿射变换，但你可能不需要计算潜在的昂贵的非线性比例或简单的导航事件的投影。 也可以将仿射变换矩阵相乘在一起，然后在一步之中将它们应用于坐标。 这对所有可能的变换不都是有效的。

这里是在 `ax.transData` 实例在基本可分离的 `Axes` 类中的定义方式。

```
self.transData = self.transScale + (self.transLimits + self.transAxes)
```

我们已经在 `Axes` 坐标中引入了上面的 `transAxes` 实例，它将轴或子图边界框的 `(0,0)` ， `(1,1)` 角映射到显示空间，所以让我们看看这两个部分。

`self.transLimits` 是从数据到轴域坐标的变换; 也就是说，它将你的视图 `xlim` 和 `ylim` 映射到轴域单位空间（然后 `transAxes` 将该单位空间用于显示空间）。 我们可以在这里看到这一点：

```
In [80]: ax = subplot(111)

In [81]: ax.set_xlim(0, 10)
Out[81]: (0, 10)

In [82]: ax.set_ylim(-1,1)
Out[82]: (-1, 1)

In [84]: ax.transLimits.transform((0,-1))
Out[84]: array([ 0.,  0.])

In [85]: ax.transLimits.transform((10,-1))
Out[85]: array([ 1.,  0.])

In [86]: ax.transLimits.transform((10,1))
Out[86]: array([ 1.,  1.])

In [87]: ax.transLimits.transform((5,0))
Out[87]: array([ 0.5,  0.5])
```

而且我们可以使用相同的反转变换，从轴域单位坐标变换回数据坐标。

```
In [90]: inv.transform((0.25, 0.25))
Out[90]: array([ 2.5, -0.5])
```

最后一个是 `self.transScale` 属性，它负责数据的可选非线性缩放，例如对数轴域。 当 `Axes` 初始化时，这只是设置为恒等变换，因为基本的 matplotlib 轴域具有线性缩放，但是当你调用对数缩放函数如 `semilogx()` 或使用 `set_xscale` 显式设置为对数时， `ax.transScale` 属性为处理非线性投影而设置。 缩放变换是相应 `xaxis` 和 `yaxis` 的 `Axis` 实例的属性。 例如，当调用 `ax.set_xscale('log')` 时， `xaxis` 会将其缩放更新为 `matplotlib.scale.LogScale` 实例。

对于不可分离的轴域， `PolarAxes` ，还有一个要考虑的部分，投影变换。 `matplotlib.projections.polar.PolarAxes` 的 `transData` 类似于典型的可分离 matplotlib 轴域，带有一个额外的部分， `transProjection` ：

```
self.transData = self.transScale + self.transProjection + \
    (self.transProjectionAffine + self.transAxes)
```

`transProjection` 将来自空间的投影，例如，地图数据的纬度和经度，或极坐标数据的半径和极角，处理为可分离的笛卡尔坐标系。 在 `matplotlib.projections` 包中有几个投影示例，深入了解的最好方法是打开这些包的源代码，看看如何自己制作它，因为 matplotlib 支持可扩展的轴域和投影。 Michael Droettboom 提供了一个创建一个锤投影轴域的很好的教程示例；请参阅 api 示例代码： `custom_projection_example.py` 。

路径教程

> 原文：Path Tutorial
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

位于所有 `matplotlib.patch` 对象底层的对象是 `Path`，它支持 `moveto`，`lineto`，`curveto` 命令的标准几个，来绘制由线段和样条组成的简单和复合轮廓。路径由 `(x,y)` 顶点的 `(N,2)` 数组，以及路径代码的长度为 N 的数组实例化。例如，为了绘制 `(0,0)` 到 `(1,1)` 的单位矩形，我们可以使用这个代码：

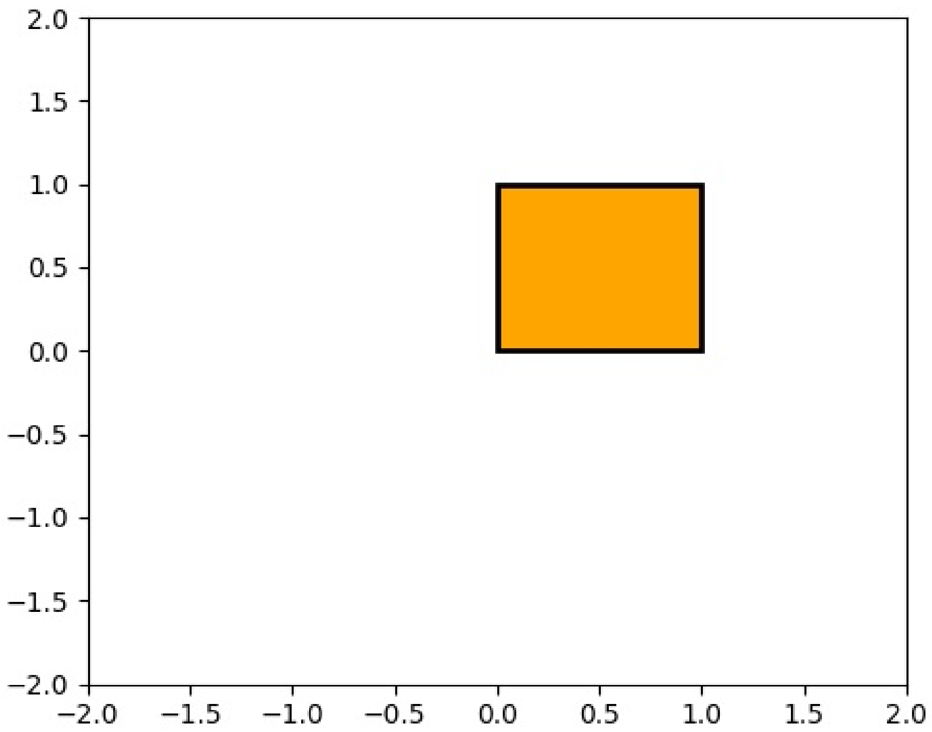
```
import matplotlib.pyplot as plt
from matplotlib.path import Path
import matplotlib.patches as patches

verts = [
    (0., 0.), # left, bottom
    (0., 1.), # left, top
    (1., 1.), # right, top
    (1., 0.), # right, bottom
    (0., 0.), # ignored
    ]

codes = [Path.MOVETO,
         Path.LINETO,
         Path.LINETO,
         Path.LINETO,
         Path.CLOSEPOLY,
         ]

path = Path(verts, codes)

fig = plt.figure()
ax = fig.add_subplot(111)
patch = patches.PathPatch(path, facecolor='orange', lw=2)
ax.add_patch(patch)
ax.set_xlim(-2,2)
ax.set_ylim(-2,2)
plt.show()
```



下面的路径代码会被接受：

代码	顶点	描述
`STOP`	1（被忽略）	标志整个路径终点的标记（当前不需要或已忽略）
`MOVETO`	1	提起笔并移动到指定顶点
`LINETO`	1	从当前位置向指定顶点画线
`CURVE3`	2（一个控制点，一个终点）	从当前位置，以给定控制点向给定端点画贝塞尔曲线
`CURVE4`	3（两个控制点，一个终点）	从当前位置，以给定控制点向给定端点画三次贝塞尔曲线
`CLOSEPOLY`	1（点自身被忽略）	向当前折线的起点画线

贝塞尔示例

一些路径组件需要以多个顶点来指定：例如 `CURVE3` 是具有一个控制点和一个端点的贝塞尔曲线，`CURVE4` 具有用做两个控制点和端点的三个顶点。下面的示例显示了 `CURVE4` 贝塞尔曲线 - 贝塞尔曲线将包含在起始点，两个控制点和终点的凸包中：

```python
import matplotlib.pyplot as plt
from matplotlib.path import Path
import matplotlib.patches as patches

verts = [
    (0., 0.),  # P0
    (0.2, 1.), # P1
    (1., 0.8), # P2
    (0.8, 0.), # P3
    ]

codes = [Path.MOVETO,
         Path.CURVE4,
         Path.CURVE4,
         Path.CURVE4,
         ]

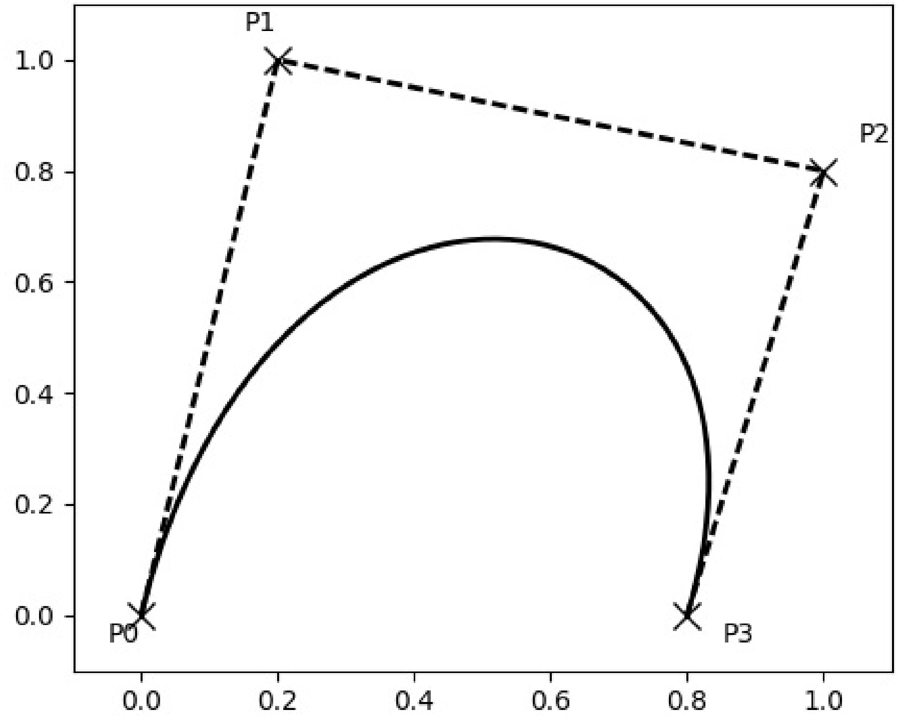
path = Path(verts, codes)

fig = plt.figure()
ax = fig.add_subplot(111)
patch = patches.PathPatch(path, facecolor='none', lw=2)
ax.add_patch(patch)

xs, ys = zip(*verts)
ax.plot(xs, ys, 'x--', lw=2, color='black', ms=10)

ax.text(-0.05, -0.05, 'P0')
ax.text(0.15, 1.05, 'P1')
ax.text(1.05, 0.85, 'P2')
ax.text(0.85, -0.05, 'P3')

ax.set_xlim(-0.1, 1.1)
ax.set_ylim(-0.1, 1.1)
plt.show()
```



复合路径

所有在 matplotlib，Rectangle，Circle，Polygon 等中的简单补丁原语都是用简单的路径实现的。通过使用复合路径，通常可以更有效地实现绘制函数，如 `hist()` 和 `bar()` ，它们创建了许多原语，例如一堆 `Rectangle` ，通常可使用复合路径来实现。 `bar` 创建一个矩形列表，而不是一个复合路径，很大程度上出于历史原因：路径代码是比较新的， `bar` 在它之前就存在。虽然我们现在可以改变它，但它会破坏旧的代码，所以如果你需要为了效率，在你自己的代码中这样做，例如，创建动画条形图，在这里我们将介绍如何创建复合路径，替换 `bar` 中的功能。

我们将通过为每个直方图的条形创建一系列矩形，来创建直方图图表：矩形宽度是条形的宽度，矩形高度是该条形中的数据点数量。首先，我们将创建一些随机的正态分布数据并计算直方图。因为 numpy 返回条形边缘而不是中心，所以下面的示例中 `bins` 的长度比 `n` 的长度大 1：

```
# histogram our data with numpy
data = np.random.randn(1000)
n, bins = np.histogram(data, 100)
```

我们现在将提取矩形的角。下面的每个 `left` ， `bottom` 等数组长度为 `len(n)` ，其中 `n` 是每个直方图条形的计数数组：

```
# get the corners of the rectangles for the histogram
left = np.array(bins[:-1])
right = np.array(bins[1:])
bottom = np.zeros(len(left))
top = bottom + n
```

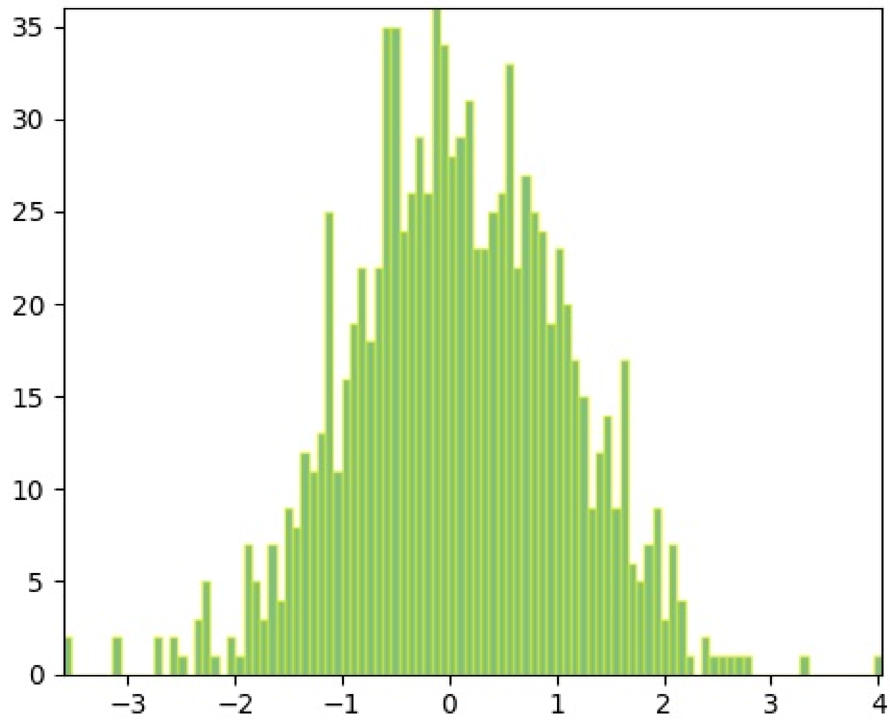
现在我们必须构造复合路径，它由每个矩形的一系列 `MOVETO` ， `LINETO` 和 `CLOSEPOLY` 组成。 对于每个矩形，我们需要 5 个顶点：一个代表 `MOVETO` ，三个代表 `LINETO` ，一个代表 `CLOSEPOLY` 。 如上表所示， `closepoly` 的顶点被忽略，但我们仍然需要它来保持代码与顶点对齐：

```
nverts = nrects*(1+3+1)
verts = np.zeros((nverts, 2))
codes = np.ones(nverts, int) * path.Path.LINETO
codes[0::5] = path.Path.MOVETO
codes[4::5] = path.Path.CLOSEPOLY
verts[0::5,0] = left
verts[0::5,1] = bottom
verts[1::5,0] = left
verts[1::5,1] = top
verts[2::5,0] = right
verts[2::5,1] = top
verts[3::5,0] = right
verts[3::5,1] = bottom
```

剩下的就是创建路径了，将其添加到 `PathPatch` ，将其添加到我们的轴域：

```
barpath = path.Path(verts, codes)
patch = patches.PathPatch(barpath, facecolor='green',
  edgecolor='yellow', alpha=0.5)
ax.add_patch(patch)
```

结果为：


35
30
25
20
15
10
5
0
−3
−2
−1
0
1
2
3
4

路径效果指南

> 原文：[Path effects guide]
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

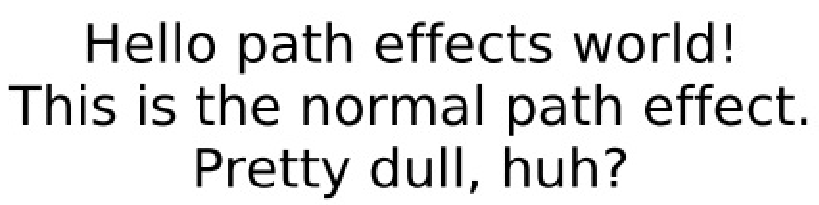
Matplotlib 的 `patheffects` 模块提供了一些功能，用于将多个绘制层次应用到任何艺术家，并可以通过路径呈现。

可以对其应用路径效果的艺术家包括 `Patch` ， `Line2D` ， `Collection` ，甚至文本。 每个艺术家的路径效果都可以通过 `set_path_effects` 方法（ `set_path_effects` ）控制，它需要一个 `AbstractPathEffect` 的可迭代实例。

最简单的路径效果是普通效果，它简单地绘制艺术家，并没有任何效果：

```
import matplotlib.pyplot as plt
import matplotlib.patheffects as path_effects

fig = plt.figure(figsize=(5, 1.5))
text = fig.text(0.5, 0.5, 'Hello path effects world!\nThis is th
e normal '
                          'path effect.\nPretty dull, huh?',
                ha='center', va='center', size=20)
text.set_path_effects([path_effects.Normal()])
plt.show()
```



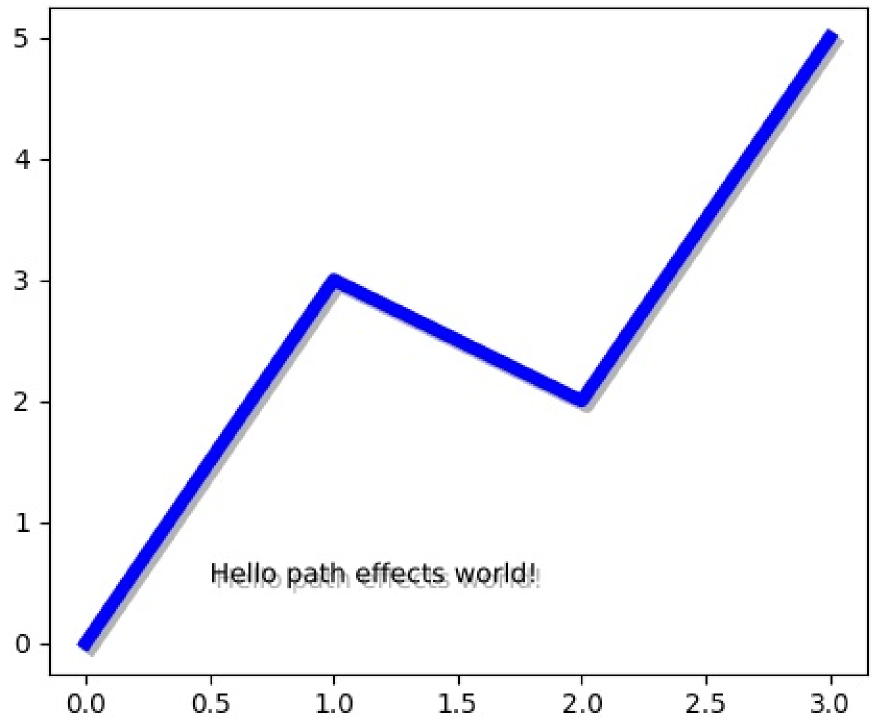
添加阴影

比正常效果更有趣的路径效果是阴影，我们可以应用于任何基于路径的艺术家。 `SimplePatchShadow` 和 `SimpleLineShadow` 类通过在基本艺术家下面绘制填充补丁或线条补丁来实现它：

```
import matplotlib.pyplot as plt
import matplotlib.patheffects as path_effects

text = plt.text(0.5, 0.5, 'Hello path effects world!',
                path_effects=[path_effects.withSimplePatchShadow
()])

plt.plot([0, 3, 2, 5], linewidth=5, color='blue',
         path_effects=[path_effects.SimpleLineShadow(),
                       path_effects.Normal()])
plt.show()
```



请注意本示例中设置路径效果的两种方法。第一个使用 `with *` 类，来包含“正常”效果之后的所需功能，而后者明确定义要绘制的两个路径效果。

让艺术家脱颖而出

使艺术家在视觉上脱颖而出的一个好方法是，在实际艺术家下面以粗体颜色绘制轮廓。 `Stroke` 路径效果使其相对简单：

```
import matplotlib.pyplot as plt
import matplotlib.patheffects as path_effects

fig = plt.figure(figsize=(7, 1))
text = fig.text(0.5, 0.5, 'This text stands out because of\n'
                          'its black border.', color='white',
                          ha='center', va='center', size=30)
text.set_path_effects([path_effects.Stroke(linewidth=3, foregrou
nd='black'),
                       path_effects.Normal()])
plt.show()
```



重要的是注意，这种效果能够工作，因为我们已经绘制两次文本路径：一次使用粗黑线，然后另一次使用原始文本路径在上面绘制。

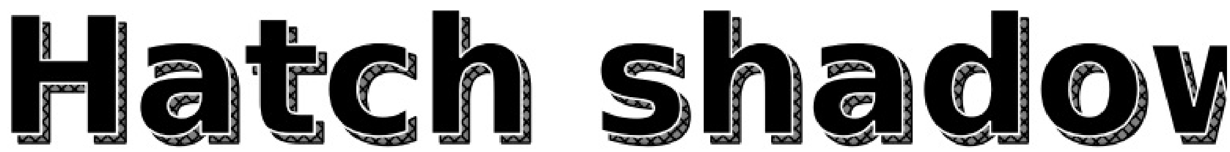
您可能已经注意到， `Stroke` 、 `SimplePatchShadow` 和 `SimpleLineShadow` 的关键字不是通常的 `Artist` 关键字（例如 `facecolor` 和 `edgecolor` 等）。这是因为使用这些路径效果，我们操作了 matplotlib 的较低层。实际上，接受的关键字是用于 `matplotlib.backend_bases.GraphicsContextBase` 实例的关键字，它们为易于创建新的后端而设计，而不是用于其用户界面。

对路径效果艺术家的更大控制

如前所述，一些路径效果的操作级别低于大多数用户操作，这意味着设置关键字（如 `facecolor` 和 `edgecolor` ）会导致 `AttributeError` 。幸运的是，有一个通用的 `PathPatchEffect` 路径效果，它创建一个具有原始路径的 `PathPatch` 类。此效果的关键字与 `PathPatch` 相同：

```
import matplotlib.pyplot as plt
import matplotlib.patheffects as path_effects

fig = plt.figure(figsize=(8, 1))
t = fig.text(0.02, 0.5, 'Hatch shadow', fontsize=75, weight=1000
, va='center')
t.set_path_effects([path_effects.PathPatchEffect(offset=(4, -4),
 hatch='xxxx',
                                                  facecolor='gra
y'),
                    path_effects.PathPatchEffect(edgecolor='whit
e', linewidth=1.1,
                                                 facecolor='blac
k')])
plt.show()
```



文本处理

引言

> 原文：Text introduction
>
> 译者：飞龙
>
>

matplotlib 具有优秀的文本支持，包括数学表达式，光栅和向量输出的 truetype 支持，任意旋转的换行分隔文本和 unicode 支持。 因为我们直接在输出文档中嵌入字体，例如 postscript 或 PDF，你在屏幕上看到的也是你在打印件中得到的。 freetype2 可产生非常漂亮，抗锯齿的字体，即使在小光栅尺寸下看起来也不错。 matplotlib 拥有自己的 `matplotlib.font_manager` ，感谢 Paul Barrett，他实现了一个跨平台，符合 W3C 标准的字体查找算法。

你可以完全控制每个文本属性（字体大小，字体重量，文本位置和颜色等），并在 `rc` 文件中设置合理的默认值。 并且对于那些对数学或科学图像感兴趣的人，matplotlib 实现了大量的 TeX 数学符号和命令，来支持你图中任何地方放置数学表达式。

基本的文本命令

> 原文：Basic text commands
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

text

在 `Axes` 的任意位置添加文本。

命令式：`matplotlib.pyplot.text`，面向对象：`matplotlib.axes.Axes.text`。

xlabel

向 x 轴添加轴标签。

命令式：`matplotlib.pyplot.xlabel`，面向对象：`matplotlib.axes.Axes.set_xlabel`。

ylabel

向 y 轴添加轴标签。

命令式：`matplotlib.pyplot.ylabel`，面向对象：`matplotlib.axes.Axes.set_ylabel`。

title

向 `Axes` 添加标题。

命令式：`matplotlib.pyplot.title`，面向对象：`matplotlib.axes.Axes.set_title`。

figtext

向 `Figure` 的任意位置添加文本。

命令式：`matplotlib.pyplot.figtext`，面向对象：`matplotlib.figure.Figure.text`。

suptitle

向`Figure`添加标题。

命令式：`matplotlib.pyplot.suptitle`，面向对象：`matplotlib.figure.Figure.suptitle`。

annotate

向`Axes`添加标注，带有可选的箭头。

命令式：`matplotlib.pyplot.annotate`，面向对象：`matplotlib.axes.Axes.annotate`。

所有这些函数创建并返回一个`matplotlib.text.Text()`实例，它可以配置各种字体和其他属性。下面的示例在实战中展示所有这些命令。

```
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt

fig = plt.figure()
fig.suptitle('bold figure suptitle', fontsize=14, fontweight='bo
ld')

ax = fig.add_subplot(111)
fig.subplots_adjust(top=0.85)
ax.set_title('axes title')

ax.set_xlabel('xlabel')
ax.set_ylabel('ylabel')

ax.text(3, 8, 'boxed italics text in data coords', style='italic'
,
        bbox={'facecolor':'red', 'alpha':0.5, 'pad':10})

ax.text(2, 6, r'an equation: $E=mc^2$', fontsize=15)

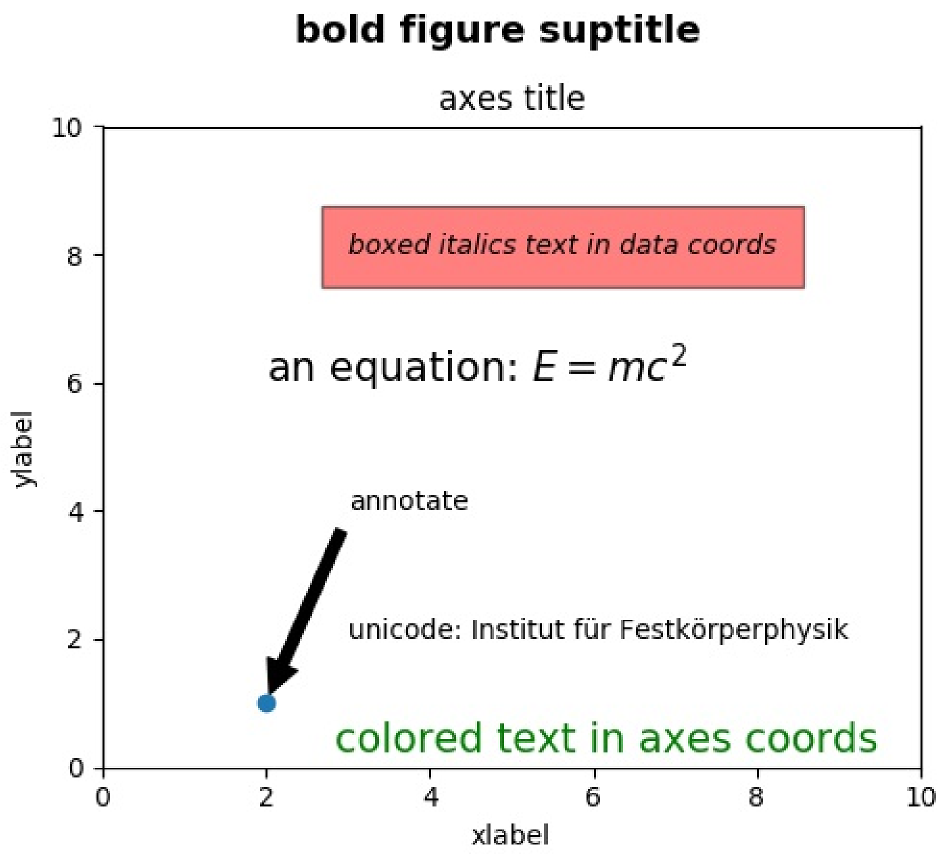
ax.text(3, 2, u'unicode: Institut f\374r Festk\366rperphysik')

ax.text(0.95, 0.01, 'colored text in axes coords',
        verticalalignment='bottom', horizontalalignment='right',
        transform=ax.transAxes,
        color='green', fontsize=15)


ax.plot([2], [1], 'o')
ax.annotate('annotate', xy=(2, 1), xytext=(3, 4),
            arrowprops=dict(facecolor='black', shrink=0.05))

ax.axis([0, 10, 0, 10])

plt.show()
```


bold figure suptitle
axes title
boxed italics text in data coords
an equation: $E = mc^2$
annotate
unicode: Institut für Festkörperphysik
colored text in axes coords
ylabel
xlabel
0
2
4
6
8
10
0
2
4
6
8
10

文本属性及布局

> 原文：Text properties and layout
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

`matplotlib.text.Text` 实例有各种属性，可以通过关键字参数配置文本命令（例如，`title()`，`xlabel()` 和 `text()`）。

属性	值类型
alpha	浮点
backgroundcolor	任何 matplotlib 颜色
bbox	rectangle prop dict plus key 'pad' which is a pad in points
clip_box	`matplotlib.transform.Bbox` 实例
clip_on	`[True / False]`
clip_path	`Path`，`Transform` 或 `Patch` 实例
color	任何 matplotlib 颜色
family	`[ 'serif' / 'sans-serif' / 'cursive' / 'fantasy`
fontproperties	`matplotlib.font_manager.FontProperties` 实例
horizontalalignment or ha	`[ 'center' / 'right' / 'left' ]`
label	任何字符串
linespacing	浮点
multialignment	`['left' / 'right' / 'center' ]`
name or fontname	字符串，例如 `['Sans' / 'Courier' / 'Helvetica' .`
picker	[`None` / 浮点 / 布尔值 / 可调用对象]`
position	`(x,y)`
rotation	[角度制的角度 / 'vertical' / 'horizontal'
size or fontsize	[点的尺寸
style or fontstyle	`[ 'normal' / 'italic' / 'oblique']`

text	字符串或任何可使用 '%s' 打印的东西
transform	`matplotlib.transform` 实例
variant	`[ 'normal' / 'small-caps' ]`
verticalalignment or va	`[ 'center' / 'top' / 'bottom' / 'baseline' ]`
visible	`[True / False]`
weight or fontweight	`[ 'normal' / 'bold' / 'heavy' / 'light' / 'ultr`
x	浮点
y	浮点
zorder	任意数值

你可以使用对齐参数 `horizontalalignment`，`verticalalignment` 和 `multialignment` 来布置文本。 `horizontalalignment` 控制文本的 x 位置参数表示文本边界框的左边，中间或右边。 `verticalalignment` 控制文本的 y 位置参数表示文本边界框的底部，中心或顶部。 `multialignment` ，仅对于换行符分隔的字符串，控制不同的行是左，中还是右对齐。 这里是一个使用 `text()` 命令显示各种对齐方式的例子。 在整个代码中使用 `transform = ax.transAxes` ，表示坐标相对于轴边界框给出，其中 `0,0` 是轴的左下角， `1,1` 是右上角。

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# build a rectangle in axes coords
left, width = .25, .5
bottom, height = .25, .5
right = left + width
top = bottom + height

fig = plt.figure()
ax = fig.add_axes([0,0,1,1])

# axes coordinates are 0,0 is bottom left and 1,1 is upper right
p = patches.Rectangle(
    (left, bottom), width, height,
    fill=False, transform=ax.transAxes, clip_on=False
    )

ax.add_patch(p)

ax.text(left, bottom, 'left top',
        horizontalalignment='left',
        verticalalignment='top',
```

```
        transform=ax.transAxes)

ax.text(left, bottom, 'left bottom',
        horizontalalignment='left',
        verticalalignment='bottom',
        transform=ax.transAxes)

ax.text(right, top, 'right bottom',
        horizontalalignment='right',
        verticalalignment='bottom',
        transform=ax.transAxes)

ax.text(right, top, 'right top',
        horizontalalignment='right',
        verticalalignment='top',
        transform=ax.transAxes)

ax.text(right, bottom, 'center top',
        horizontalalignment='center',
        verticalalignment='top',
        transform=ax.transAxes)

ax.text(left, 0.5*(bottom+top), 'right center',
        horizontalalignment='right',
        verticalalignment='center',
        rotation='vertical',
        transform=ax.transAxes)

ax.text(left, 0.5*(bottom+top), 'left center',
        horizontalalignment='left',
        verticalalignment='center',
        rotation='vertical',
        transform=ax.transAxes)

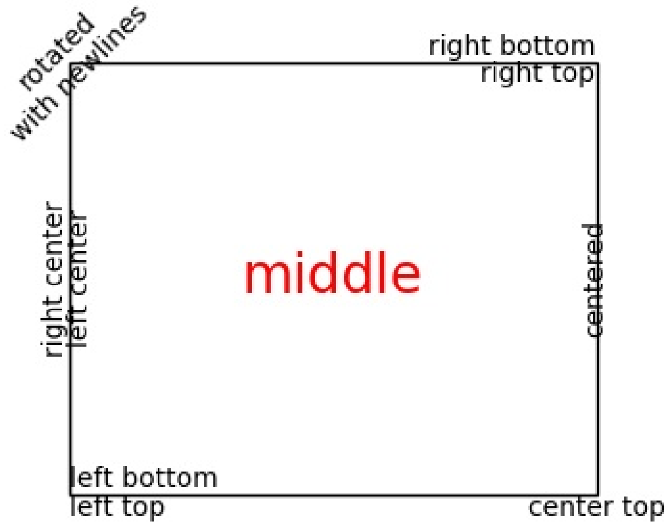
ax.text(0.5*(left+right), 0.5*(bottom+top), 'middle',
        horizontalalignment='center',
        verticalalignment='center',
        fontsize=20, color='red',
        transform=ax.transAxes)

ax.text(right, 0.5*(bottom+top), 'centered',
        horizontalalignment='center',
        verticalalignment='center',
        rotation='vertical',
        transform=ax.transAxes)

ax.text(left, top, 'rotated\nwith newlines',
        horizontalalignment='center',
        verticalalignment='center',
        rotation=45,
        transform=ax.transAxes)

ax.set_axis_off()
```

```
plt.show()
```


rotated
with newlines
right bottom
right top
right center
left center
middle
centered
left bottom
left top
center top

默认字体

> 原文：Text properties and layout
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

基本的默认字体由一系列 `rcParams` 参数控制：

`rcParam`	用法
`'font.family'`	字体名称或 `{'cursive', 'fantasy', 'monospace', 'sans', '` 列表
`'font.style'`	默认字体，例如 `'normal'` ， `'italic'`
`'font.variant'`	默认变体，例如 `'normal'` ， `'small-caps'` （未测试）
`'font.stretch'`	默认拉伸 `'normal'` ， `'condensed'` （未完成）
`'font.weight'`	字体粗细，可为整数或字符串
`'font.size'`	默认字体大小（以磅为单位）。相对字体大小（ `'large'`

字体系列别名（ `{'cursive'，'fantasy'，'monospace'，'sans'，'sans serif'，'sans-ser` ）和实际字体名称之间的映射由以下 `rcParams` 控制 ：

系列别名	映射的 `rcParam`
`'serif'`	`'font.serif'`
`'monospace'`	`'font.monospace'`
`'fantasy'`	`'font.fantasy'`
`'cursive'`	`'font.cursive'`
`{'sans', 'sans serif', 'sans-serif'}`	`'font.sans-serif'`

它是字体名称的列表。

非拉丁字形文本

从 v2.0 开始，默认字体包含许多西方字母的字形，但仍然没有覆盖 mpl 用户可能需要的所有字形。 例如，DejaVu 没有覆盖中文，韩语或日语。

要将默认字体设置为支持所需代码点的字体，请将字体名称添加到 `font.family` 或所需的别名列表前面。

```
matplotlib.rcParams['font.sans-serif'] = ['Source Han Sans TW', 'sans-serif']
```

或在 `.matplotlibrc` 文件中设置：

```
font.sans-serif: Source Han Sans TW, Ariel, sans-serif
```

要控制每个艺术家使用的字体，使用上面记录的 `'name'`，`'fontname'` 或 `'fontproperties'` 关键字参数。

在 linux 上，`fc-list` 是用于发现字体名称的实用工具；例如

```
$ fc-list :lang=zh family
Noto to Sans Mono CJK TC,Noto Sans Mono CJK TC Bold
Noto Sans CJK TC,Noto Sans CJK TC Medium
Noto Sans CJK TC,Noto Sans CJK TC DemiLight
Noto Sans CJK KR,Noto Sans CJK KR Black
Noto Sans CJK TC,Noto Sans CJK TC Black
Noto Sans Mono CJK TC,Noto Sans Mono CJK TC Regular
Noto Sans CJK SC,Noto Sans CJK SC Light
```

列出了所有支持中文的字体。

标注

> 原文：Annotation
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

基本标注

使用 `text()` 会将文本放置在轴域的任意位置。 文本的一个常见用例是标注绘图的某些特征，而 `annotate()` 方法提供辅助函数，使标注变得容易。 在标注中，有两个要考虑的点：由参数 `xy` 表示的标注位置和 `xytext` 的文本位置。 这两个参数都是 `(x, y)` 元组。

```
import numpy as np
import matplotlib.pyplot as plt

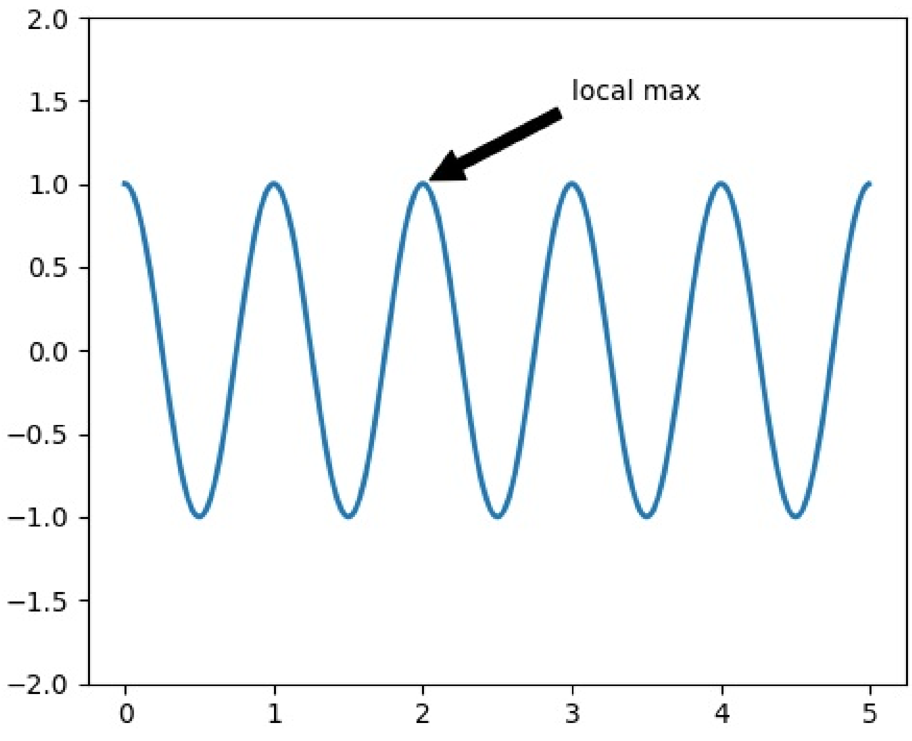
fig = plt.figure()
ax = fig.add_subplot(111)

t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2*np.pi*t)
line, = ax.plot(t, s, lw=2)

ax.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )

ax.set_ylim(-2,2)
plt.show()
```

源代码



在该示例中， `xy` （箭头尖端）和 `xytext` 位置（文本位置）都以数据坐标为单位。 有多种可以选择的其他坐标系 - 你可以使用 `xycoords` 和 `textcoords` 以及下列字符串之一（默认为 `data` ）指定 `xy` 和 `xytext` 的坐标系。

| 参数 | 坐标系 || `'figure points'` | 距离图形左下角的点数量 || `'figure pixels'` | 距离图形左下角的像素数量 || `'figure fraction'` | 0,0 是图形左下角，1,1 是右上角 || `'axes points'` | 距离轴域左下角的点数量 || `'axes pixels'` | 距离轴域左下角的像素数量 || `'axes fraction'` | 0,0 是轴域左下角，1,1 是右上角 || `'data'` | 使用轴域数据坐标系 |

例如将文本以轴域小数坐标系来放置，我们可以：

```
ax.annotate('local max', xy=(3, 1),  xycoords='data',
            xytext=(0.8, 0.95), textcoords='axes fraction',
            arrowprops=dict(facecolor='black', shrink=0.05),
            horizontalalignment='right', verticalalignment='top'
,
            )
```

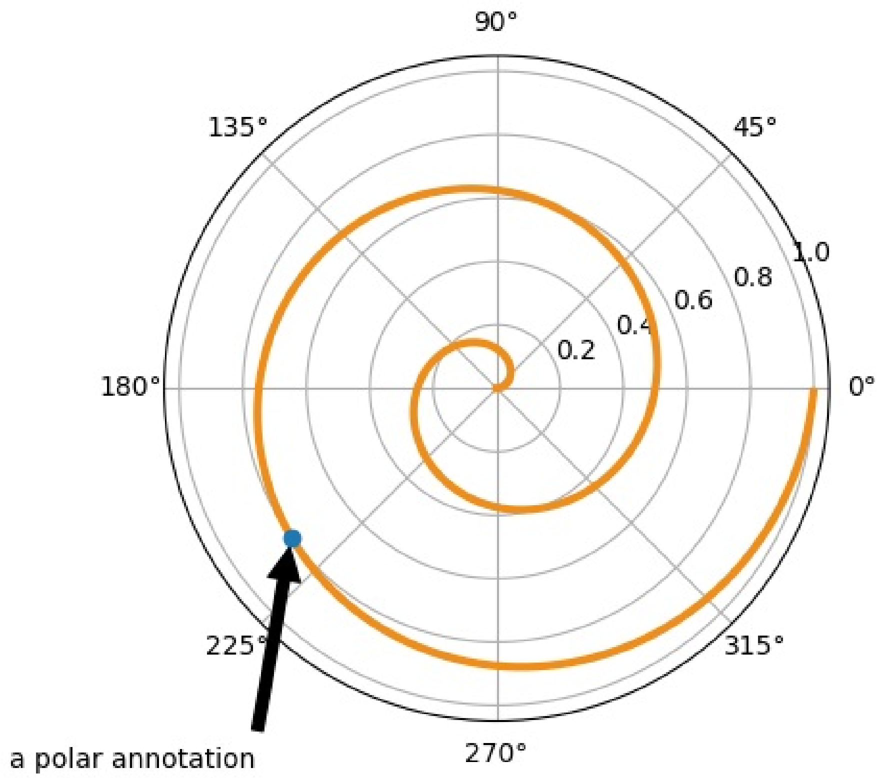
对于物理坐标系（点或像素），原点是图形或轴的左下角。

或者，你可以通过在可选关键字参数 `arrowprops` 中提供箭头属性字典来绘制从文本到注释点的箭头。

`arrowprops` 键	描述
`width`	箭头宽度，以点为单位
`frac`	箭头头部所占据的比例
`headwidth`	箭头的底部的宽度，以点为单位
`shrink`	移动提示，并使其离注释点和文本一些距离
`**kwargs`	`matplotlib.patches.Polygon` 的任何键，例如 `facecolor`

在下面的示例中， `xy` 点是原始坐标（ `xycoords` 默认为 `'data'` ）。 对于极坐标轴，它在 `(theta, radius)` 空间中。 此示例中的文本放置在图形小数坐标系中。 `matplotlib.text.Text` 关键字 `args` ，例如 `horizontalalignment` ， `verticalalignment` 和 `fontsize` ，从 `annotate` 传给 `Text` 实例。

源代码



注释（包括花式箭头）的所有高上大的内容的更多信息，请参阅高级标注和 `pylab_examples` 示例代码： `annotation_demo.py` 。

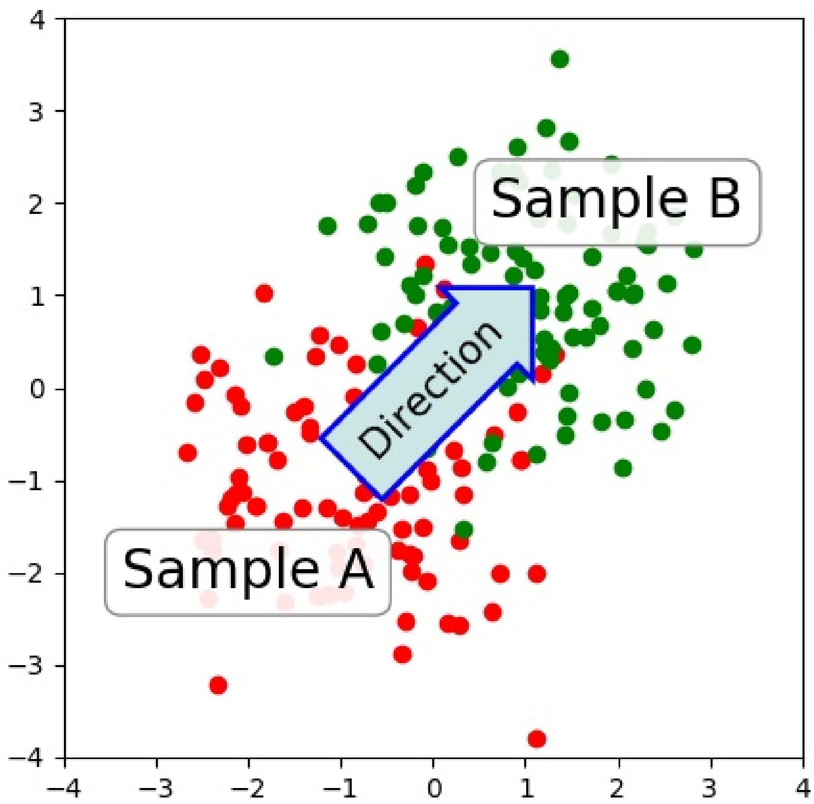
不要继续，除非你已经阅读了基本标注， `text()` 和 `annotate()` 。

高级标注

使用框和文本来标注

让我们以一个简单的例子来开始。

源代码



在 `pyplot` 模块（或 `Axes` 类的 `text` 方法）中的 `text()` 函数接受 `bbox` 关键字参数，并且在提供时，在文本周围绘制一个框。

与文本相关联的补丁对象可以通过以下方式访问：

```
bb = t.get_bbox_patch()
```

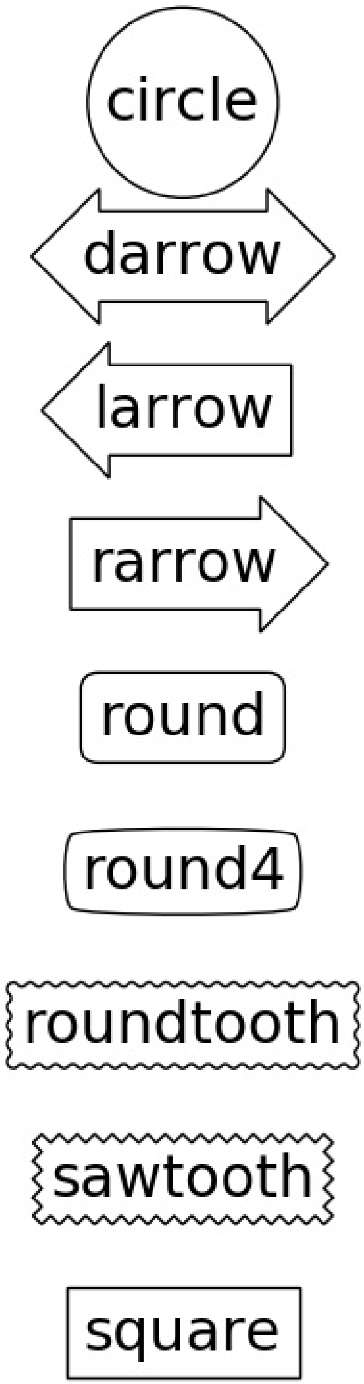
返回值是 `FancyBboxPatch` 的一个实例，并且补丁属性（如 `facecolor` ， `edgewidth` 等）可以像平常一样访问和修改。 为了更改框的形状，请使用 `set_boxstyle` 方法。

```
bb.set_boxstyle("rarrow", pad=0.6)
```

该参数是框样式的名称与其作为关键字参数的属性。 目前，实现了以下框样式。

类	名称	属性
Circle	`circle`	`pad=0.3`
DArrow	`darrow`	`pad=0.3`
LArrow	`larrow`	`pad=0.3`
RArrow	`rarrow`	`pad=0.3`
Round	`round`	`pad=0.3,rounding_size=None`
Round4	`round4`	`pad=0.3,rounding_size=None`
Roundtooth	`roundtooth`	`pad=0.3,tooth_size=None`
Sawtooth	`sawtooth`	`pad=0.3,tooth_size=None`
Square	`square`	`pad=0.3`

源代码



注意，属性参数可以在样式名称中用逗号分隔（在初始化文本实例时，此形式可以用作 `bbox` 参数的 `boxstyle` 的值）。

```
bb.set_boxstyle("rarrow,pad=0.6")
```

使用箭头来标注

`pyplot` 模块（或 `Axes` 类的 `annotate` 方法）中的 `annotate()` 函数用于绘制连接图上两点的箭头。

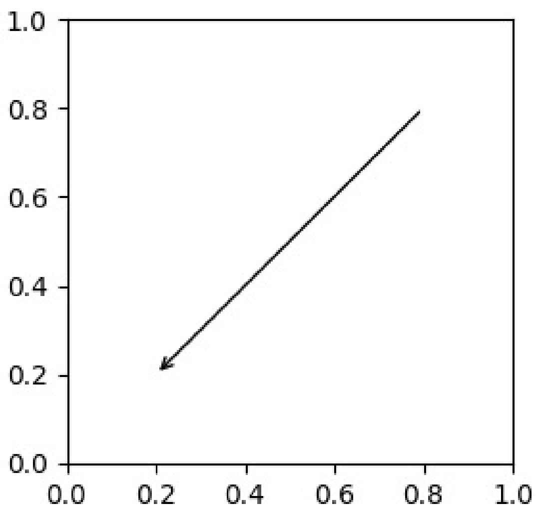
```
ax.annotate("Annotation",
            xy=(x1, y1), xycoords='data',
            xytext=(x2, y2), textcoords='offset points',
            )
```

这会使用 `textcoords` 中提供的， `xytext` 处的文本标注提供坐标（ `xycoords` ）中的 `xy` 处的点。 通常，数据坐标中规定了标注点，偏移点中规定了标注文本。 请参阅 `annotate()` 了解可用的坐标系。

连接两个点（ `xy` 和 `xytext` ）的箭头可以通过指定 `arrowprops` 参数可选地绘制。 为了仅绘制箭头，请使用空字符串作为第一个参数。

```
ax.annotate("",
            xy=(0.2, 0.2), xycoords='data',
            xytext=(0.8, 0.8), textcoords='data',
            arrowprops=dict(arrowstyle="->",
                            connectionstyle="arc3"),
            )
```

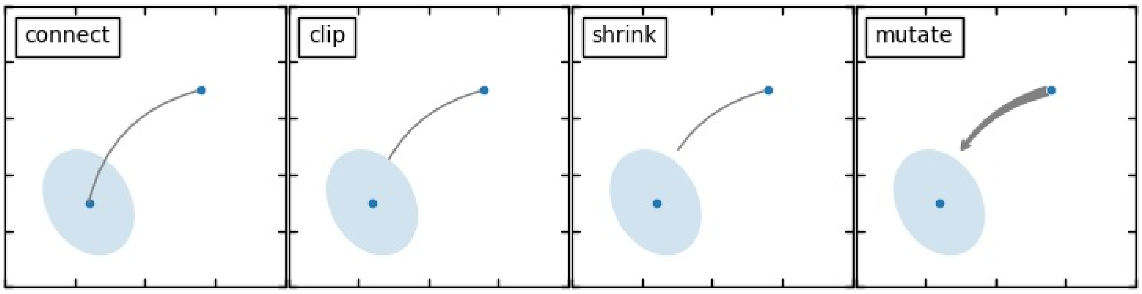
源代码



箭头的绘制需要几个步骤。

- 创建两个点之间的连接路径。 这由 `connectionstyle` 键值控制。
- 如果提供了补丁对象（ `patchA` 和 `patchB` ），则会剪切路径以避开该补丁。
- 路径进一步由提供的像素总量来缩小（ `shirnkA` & `shrinkB` ）
- 路径转换为箭头补丁，由 `arrowstyle` 键值控制。

源代码



两个点之间的连接路径的创建由 `connectionstyle` 键控制，并且可用以下样式。

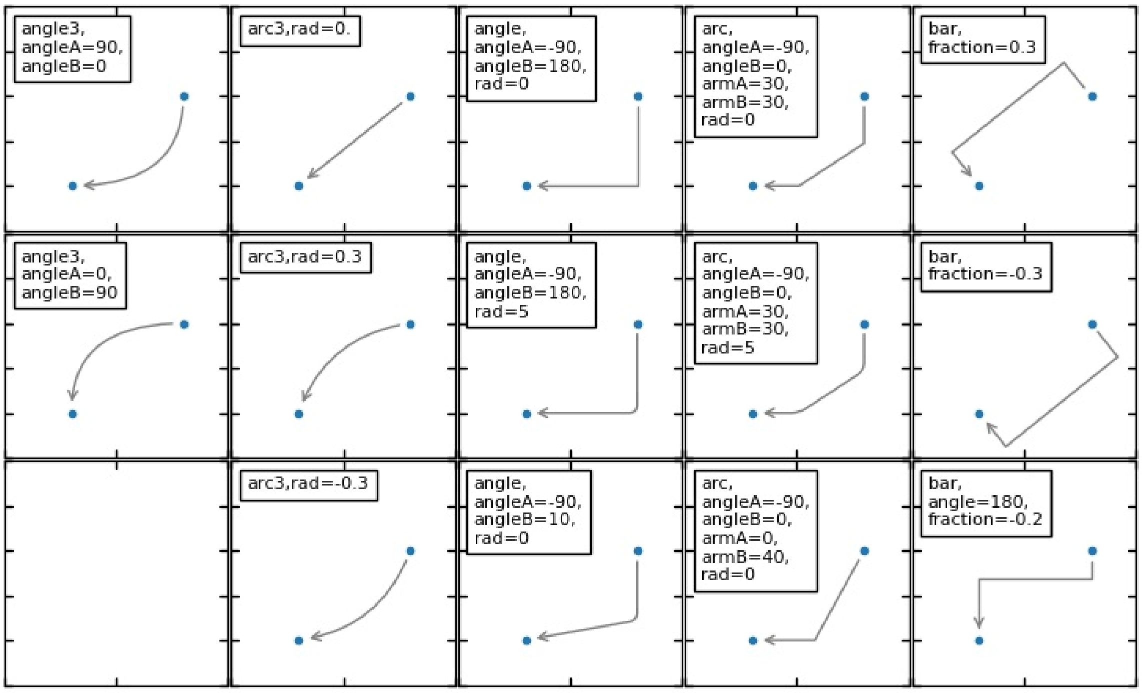
名称	属性
angle	`angleA=90,angleB=0,rad=0.0`
angle3	`angleA=90,angleB=0`
arc	`angleA=0,angleB=0,armA=None,armB=None,rad=0.0`
arc3	`rad=0.0`
bar	`armA=0.0,armB=0.0,fraction=0.3,angle=None`

注意， `angle3` 和 `arc3` 中的 `3` 意味着所得到的路径是二次样条段（三个控制点）。如下面将讨论的，当连接路径是二次样条时，可以使用一些箭头样式选项。

每个连接样式的行为在下面的示例中（有限地）演示。（警告：条形样式的行为当前未定义好，将来可能会更改）。

源代码

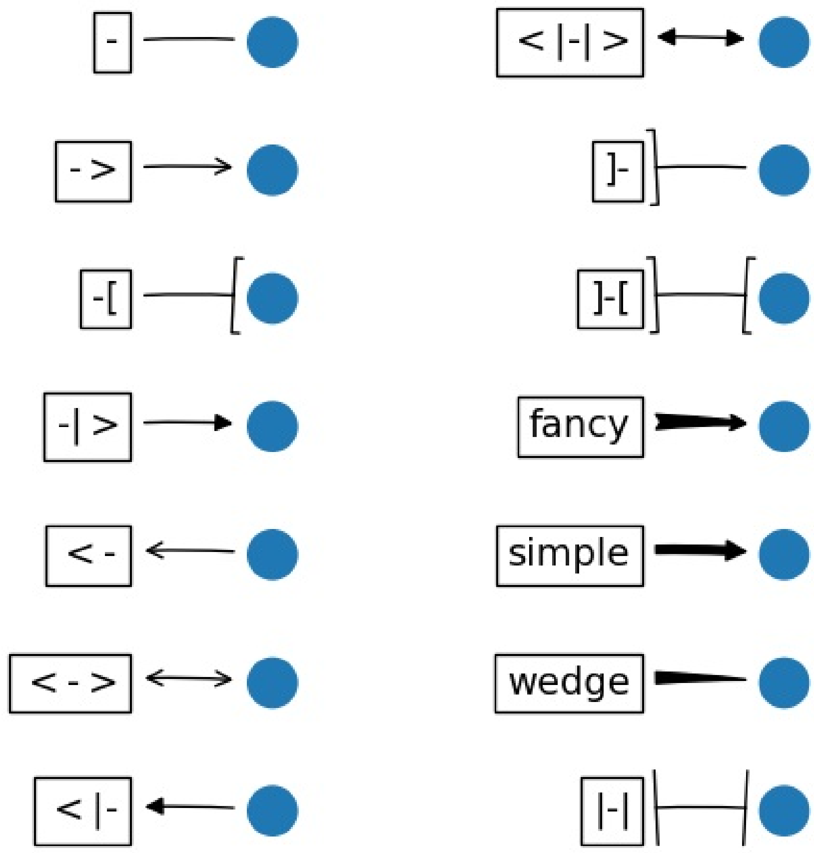
源代码



然后根据给定的箭头样式将连接路径（在剪切和收缩之后）变换为箭头补丁。

名称	属性	
-	`None`	
->	`head_length=0.4,head_width=0.2`	
-[	`widthB=1.0,lengthB=0.2,angleB=None`	
|-|	`widthA=1.0,widthB=1.0`	
-|>	`head_length=0.4,head_width=0.2`	
<-	`head_length=0.4,head_width=0.2`	
<->	`head_length=0.4,head_width=0.2`	
<|-	`head_length=0.4,head_width=0.2`	
<	-|>	`head_ler`
fancy	`head_length=0.4,head_width=0.4,tail_width=0.4`	
simple	`head_length=0.5,head_width=0.5,tail_width=0.2`	
wedge	`tail_width=0.3,shrink_factor=0.5`	

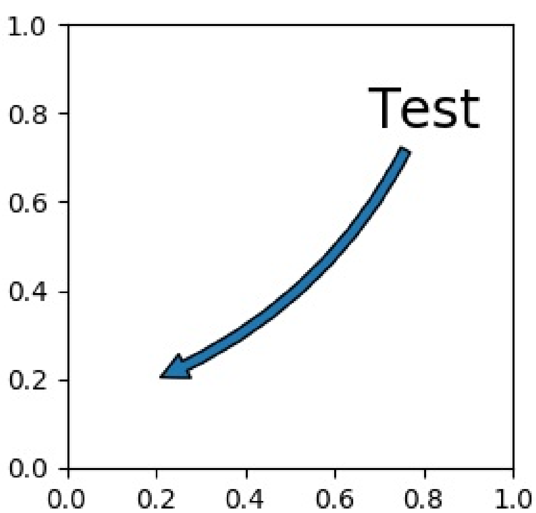
源代码



一些箭头仅适用于生成二次样条线段的连接样式。他们是 `fancy` ，`simple` ，`wedge` 。对于这些箭头样式，必须使用 `angle3` 或 `arc3` 连接样式。

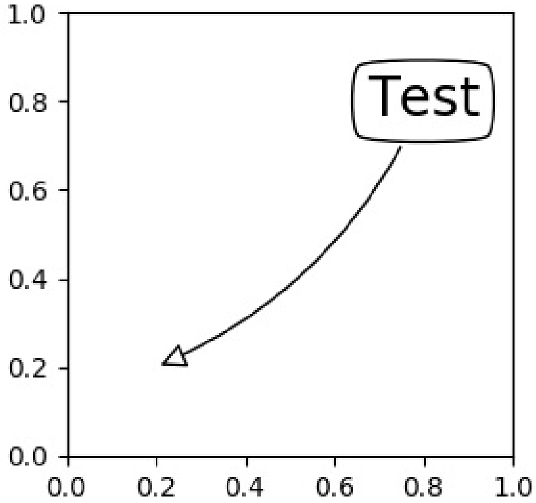
如果提供了标注字符串，则 `patchA` 默认设置为文本的 `bbox` 补丁。

源代码



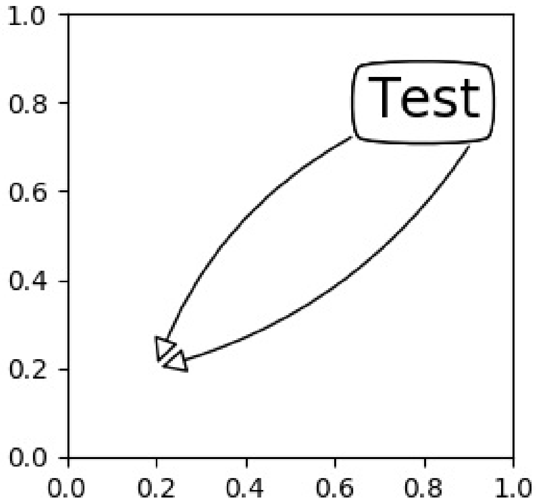
与 `text` 命令一样，可以使用 `bbox` 参数来绘制文本周围的框。

源代码



默认情况下，起点设置为文本范围的中心。可以使用`relpos`键值进行调整。这些值根据文本的范围进行归一化。例如，`(0,0)`表示左下角，`(1,1)`表示右上角。

源代码

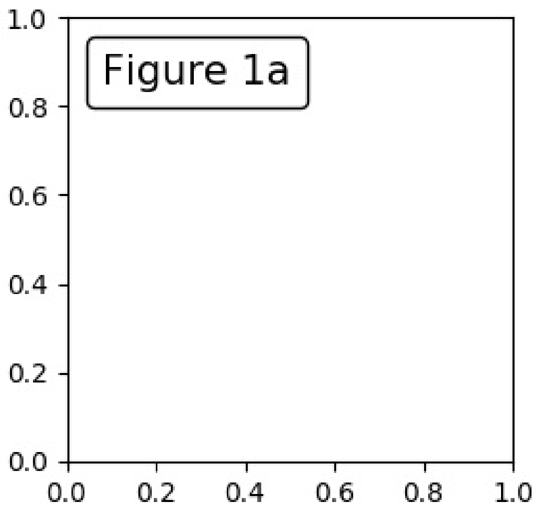


将艺术家放置在轴域的锚定位置

有一类艺术家可以放置在轴域的锚定位置。一个常见的例子是图例。这种类型的艺术家可以使用`OffsetBox`类创建。`mpl_toolkits.axes_grid.anchored_artists`中有几个预定义类。

```
from mpl_toolkits.axes_grid.anchored_artists import AnchoredText
at = AnchoredText("Figure 1a",
                  prop=dict(size=8), frameon=True,
                  loc=2,
                  )
at.patch.set_boxstyle("round,pad=0.,rounding_size=0.2")
ax.add_artist(at)
```

源代码



`loc` 关键字与 `legend` 命令中含义相同。

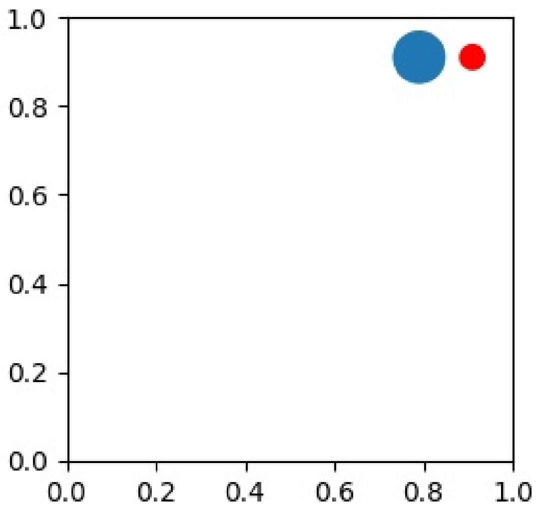
一个简单的应用是当艺术家（或艺术家的集合）的像素大小在创建时已知。例如，如果要绘制一个固定大小为 20 像素 ×20 像素（半径为 10 像素）的圆，则可以使用 `AnchoredDrawingArea` 。实例使用绘图区域的大小创建（以像素为单位）。用户可以在绘图区任意添加艺术家。注意，添加到绘图区域的艺术家的范围与绘制区域本身的位置无关，只和初始大小有关。

```
from mpl_toolkits.axes_grid.anchored_artists import AnchoredDraw
ingArea

ada = AnchoredDrawingArea(20, 20, 0, 0,
                          loc=1, pad=0., frameon=False)
p1 = Circle((10, 10), 10)
ada.drawing_area.add_artist(p1)
p2 = Circle((30, 10), 5, fc="r")
ada.drawing_area.add_artist(p2)
```

添加到绘图区域的艺术家不应该具有变换集（它们将被重写），并且那些艺术家的尺寸被解释为像素坐标，即，上述示例中的圆的半径分别是 10 像素和 5 像素。

源代码



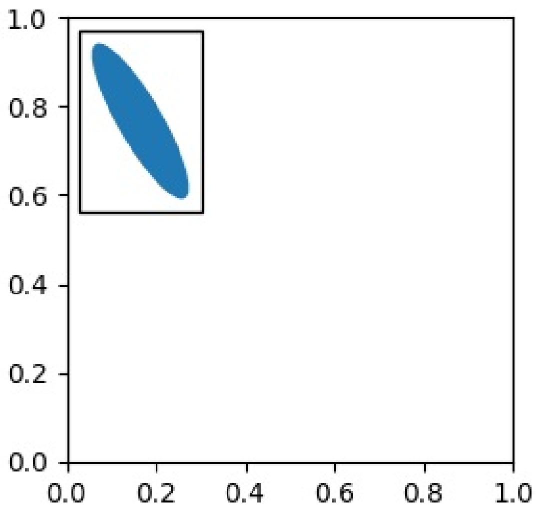
有时，你想让你的艺术家按数据坐标（或其他坐标，而不是画布像素）缩放。你可以使用 `AnchoredAuxTransformBox` 类。这类似于 `AnchoredDrawingArea` ，除了艺术家的范围在绘制时由指定的变换确定。

```
from mpl_toolkits.axes_grid.anchored_artists import AnchoredAuxT
ransformBox

box = AnchoredAuxTransformBox(ax.transData, loc=2)
el = Ellipse((0,0), width=0.1, height=0.4, angle=30) # in data c
oordinates!
box.drawing_area.add_artist(el)
```

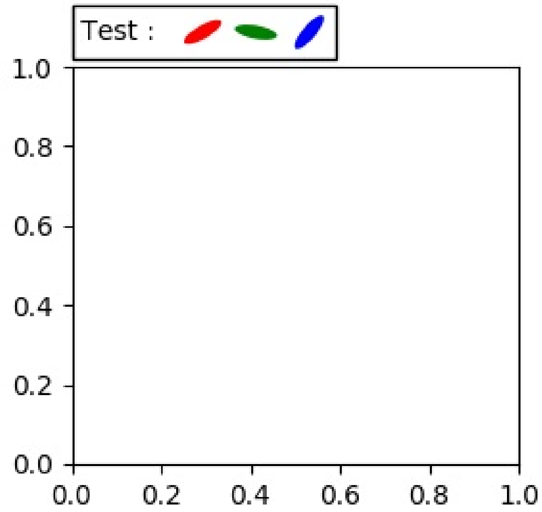
上述示例中的椭圆具有在数据坐标中对应于 0.1 和 0.4 的宽度和高度，并且当轴域的视图限制改变时将自动缩放。

源代码



如图例所示，可以设置 `bbox_to_anchor` 参数。使用 `HPacker` 和 `VPacker`，你可以像图例中一样排列艺术家（事实上，这是图例的创建方式）。

源代码



请注意，与图例不同，默认情况下，`bbox_transform` 设置为 `IdentityTransform`。

使用复杂坐标来标注

matplotlib 中的标注支持标注文本中描述的几种类型的坐标。对于想要更多控制的高级用户，它支持几个其他选项。

1. `Transform` 实例，例如：

   ```
   ax.annotate("Test", xy=(0.5, 0.5), xycoords=ax.transAxes)
   ```

   相当于：

   ```
   ax.annotate("Test", xy=(0.5, 0.5), xycoords="axes fraction")
   ```

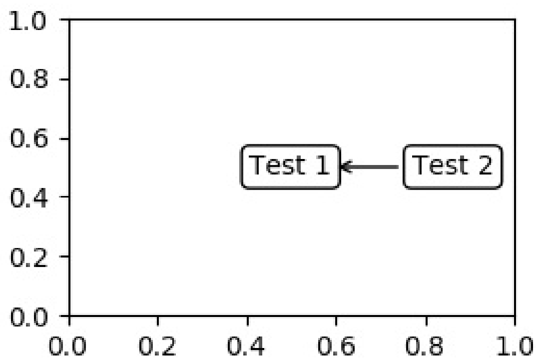
   使用它，你可以在其他轴域内标注一个点：

   ```
   ax1, ax2 = subplot(121), subplot(122)
   ax2.annotate("Test", xy=(0.5, 0.5), xycoords=ax1.transData,
                xytext=(0.5, 0.5), textcoords=ax2.transData,
                arrowprops=dict(arrowstyle="->"))
   ```

2. `Artist` 实例。`xy` 值（或 `xytext`）被解释为艺术家的 `bbox`（`get_window_extent` 的返回值）的小数坐标。

```
an1 = ax.annotate("Test 1", xy=(0.5, 0.5), xycoords="data",
                  va="center", ha="center",
                  bbox=dict(boxstyle="round", fc="w"))
an2 = ax.annotate("Test 2", xy=(1, 0.5), xycoords=an1, # (1,
0.5) of the an1's bbox
                  xytext=(30,0), textcoords="offset points",
                  va="center", ha="left",
                  bbox=dict(boxstyle="round", fc="w"),
                  arrowprops=dict(arrowstyle="->"))
```

源代码



请注意，你的责任是在绘制 `an2` 之前确定坐标艺术家（上例中的 `an1` ）的范围。 在大多数情况下，这意味着 `an2` 需要晚于 `an1` 。

3. 一个返回 `BboxBase` 或 `Transform` 的实例的可调用对象。 如果返回一个变换，它与 1 相同，如果返回 `bbox` ，它与 2 相同。可调用对象应该接受 `renderer` 实例的单个参数。 例如，以下两个命令产生相同的结果：

```
an2 = ax.annotate("Test 2", xy=(1, 0.5), xycoords=an1,
                  xytext=(30,0), textcoords="offset points")
an2 = ax.annotate("Test 2", xy=(1, 0.5), xycoords=an1.get_wi
ndow_extent,
                  xytext=(30,0), textcoords="offset points")
```

4. 指定二元坐标的元组。 第一项用于 `x` 坐标，第二项用于 `y` 坐标。 例如，

```
annotate("Test", xy=(0.5, 1), xycoords=("data", "axes fracti
on"))
```

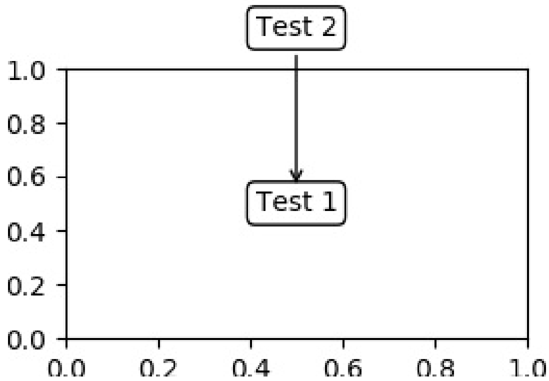
0.5 的单位是数据坐标，1 的单位是归一化轴域坐标。 你可以像使用元组一样使用艺术家或变换。 例如，

```
import matplotlib.pyplot as plt

plt.figure(figsize=(3,2))
ax=plt.axes([0.1, 0.1, 0.8, 0.7])
an1 = ax.annotate("Test 1", xy=(0.5, 0.5), xycoords="data",
                  va="center", ha="center",
                  bbox=dict(boxstyle="round", fc="w"))

an2 = ax.annotate("Test 2", xy=(0.5, 1.), xycoords=an1,
                  xytext=(0.5,1.1), textcoords=(an1, "axes f
raction"),
                  va="bottom", ha="center",
                  bbox=dict(boxstyle="round", fc="w"),
                  arrowprops=dict(arrowstyle="->"))
plt.show()
```

源代码

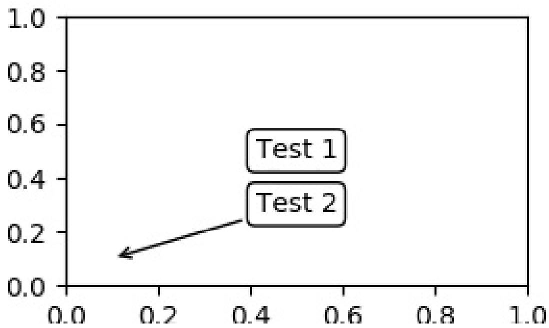


5. 有时，您希望您的注释带有一些“偏移点”，不是距离注释点，而是距离某些其他点。 `OffsetFrom` 是这种情况下的辅助类。

```
import matplotlib.pyplot as plt

plt.figure(figsize=(3,2))
ax=plt.axes([0.1, 0.1, 0.8, 0.7])
an1 = ax.annotate("Test 1", xy=(0.5, 0.5), xycoords="data",
                  va="center", ha="center",
                  bbox=dict(boxstyle="round", fc="w"))

from matplotlib.text import OffsetFrom
offset_from = OffsetFrom(an1, (0.5, 0))
an2 = ax.annotate("Test 2", xy=(0.1, 0.1), xycoords="data",
                  xytext=(0, -10), textcoords=offset_from,
                  # xytext is offset points from "xy=(0.5, 0
), xycoords=an1"
                  va="top", ha="center",
                  bbox=dict(boxstyle="round", fc="w"),
                  arrowprops=dict(arrowstyle="->"))
plt.show()
```



你可以参考这个链接：`pylab_examples example code: annotation_demo3.py.`。

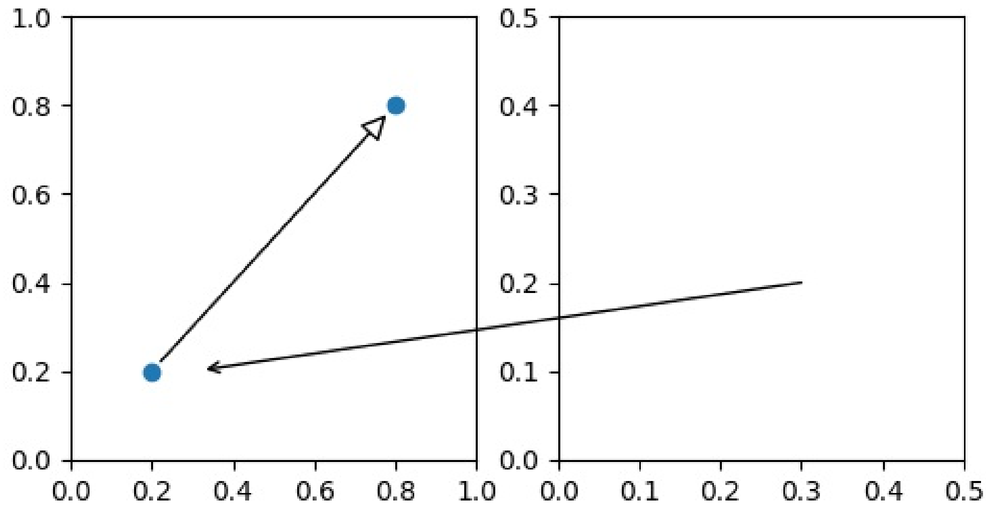
使用 `ConnectorPatch`

`ConnectorPatch` 类似于没有文本的标注。虽然在大多数情况下建议使用标注函数，但是当您想在不同的轴上连接点时，`ConnectorPatch` 很有用。

```
from matplotlib.patches import ConnectionPatch
xy = (0.2, 0.2)
con = ConnectionPatch(xyA=xy, xyB=xy, coordsA="data", coordsB="d
ata",
                      axesA=ax1, axesB=ax2)
ax2.add_artist(con)
```

上述代码连接了 `ax1` 中数据坐标的 `xy` 点，与 `ax2` 中数据坐标的 `xy` 点。这是个简单的例子。

源代码



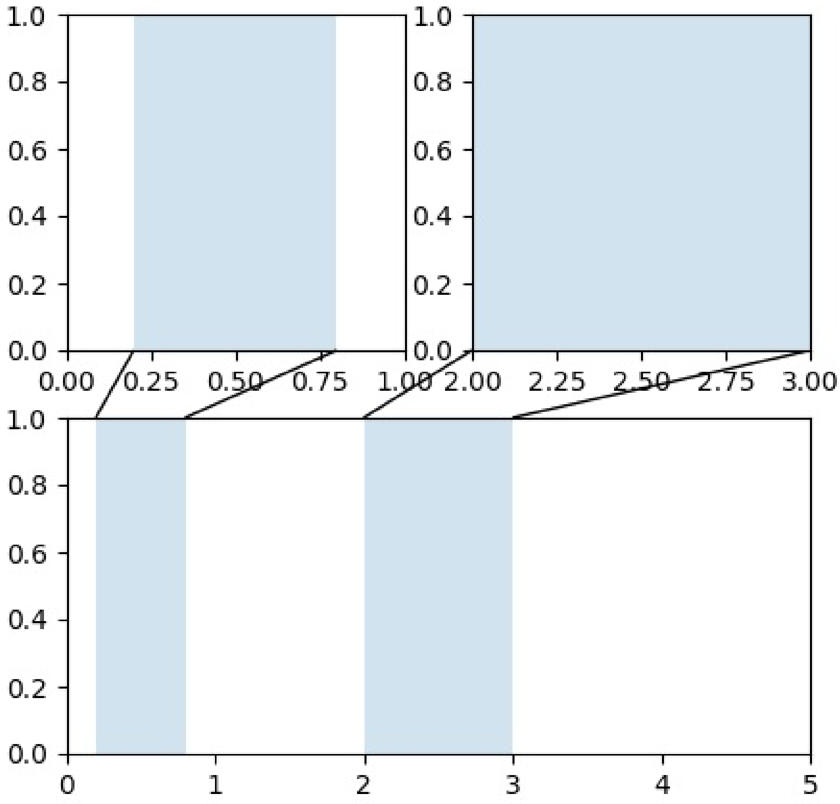
虽然 `ConnectorPatch` 实例可以添加到任何轴，但您可能需要将其添加到绘图顺序中最新的轴，以防止与其他轴重叠。

高级话题

轴域之间的缩放效果

`mpl_toolkits.axes_grid.inset_locator` 定义了一些补丁类，用于互连两个轴域。理解代码需要一些 mpl 转换如何工作的知识。但是，利用它的方式很直接。

源代码



定义自定义盒样式

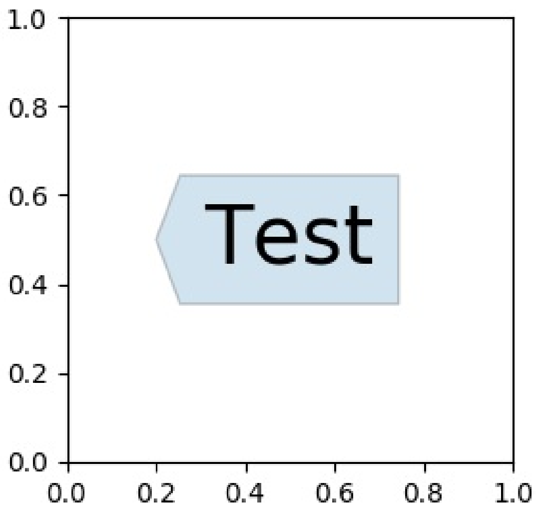
你可以使用自定义盒样式，`boxstyle` 的值可以为如下形式的可调用对象：

```
def __call__(self, x0, y0, width, height, mutation_size,
             aspect_ratio=1.):
    """
    Given the location and size of the box, return the path of
    the box around it.

      - *x0*, *y0*, *width*, *height* : location and size of the
 box
      - *mutation_size* : a reference scale for the mutation.
      - *aspect_ratio* : aspect-ratio for the mutation.
    """
    path = ...
    return path
```

这里是个复杂的例子：

源代码



但是，推荐你从`matplotlib.patches.BoxStyle._Base`派生，像这样：

```
from matplotlib.path import Path
from matplotlib.patches import BoxStyle
import matplotlib.pyplot as plt

# we may derive from matplotlib.patches.BoxStyle._Base class.
# You need to override transmute method in this case.

class MyStyle(BoxStyle._Base):
    """
    A simple box.
    """

    def __init__(self, pad=0.3):
        """
        The arguments need to be floating numbers and need to ha
ve
        default values.

         *pad*
            amount of padding
        """

        self.pad = pad
        super(MyStyle, self).__init__()

    def transmute(self, x0, y0, width, height, mutation_size):
        """
        Given the location and size of the box, return the path
of
        the box around it.

         - *x0*, *y0*, *width*, *height* : location and size of
the box
         - *mutation_size* : a reference scale for the mutation.
```

```
        Often, the *mutation_size* is the font size of the text.
        You don't need to worry about the rotation as it is
        automatically taken care of.
        """

        # padding
        pad = mutation_size * self.pad

        # width and height with padding added.
        width, height = width + 2.*pad, \
                        height + 2.*pad,

        # boundary of the padded box
        x0, y0 = x0-pad, y0-pad,
        x1, y1 = x0+width, y0 + height

        cp = [(x0, y0),
              (x1, y0), (x1, y1), (x0, y1),
              (x0-pad, (y0+y1)/2.), (x0, y0),
              (x0, y0)]

        com = [Path.MOVETO,
               Path.LINETO, Path.LINETO, Path.LINETO,
               Path.LINETO, Path.LINETO,
               Path.CLOSEPOLY]

        path = Path(cp, com)

        return path

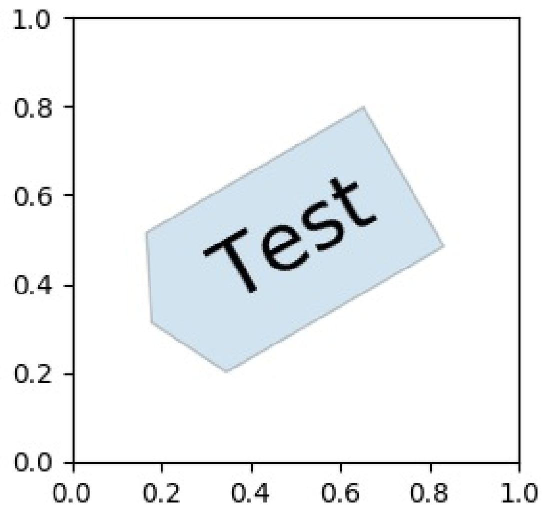

# register the custom style
BoxStyle._style_list["angled"] = MyStyle

plt.figure(1, figsize=(3,3))
ax = plt.subplot(111)
ax.text(0.5, 0.5, "Test", size=30, va="center", ha="center", rot
ation=30,
        bbox=dict(boxstyle="angled,pad=0.5", alpha=0.2))

del BoxStyle._style_list["angled"]

plt.show()
```

源代码



与之类似，您可以定义一个自定义的 `ConnectionStyle` 和一个自定义的 `ArrowStyle` 。请参阅 `lib/matplotlib/patches.py` 的源代码，并查看每个样式类是如何定义的。

编写数学表达式

> 原文：Writing mathematical expressions
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

你可以在任何 matplotlib 文本字符串中使用子 TeX 标记，将它放在一对美元符号（`$`）内。

注意，你不需要安装 TeX，因为 matplotlib 提供了自己的 TeX 表达式解析器，布局引擎和字体。布局引擎是 Donald Knuth 的 TeX 中的布局算法的一种相当直接的适配版，所以质量是相当不错的（matplotlib 还为那些想要调用 TeX 生成文本的人提供一个 `usetex` 选项（参见使用 LaTeX 渲染文本）。

任何文本元素都可以使用数学文本。你应该使用原始字符串（在引号前面加一个 `'r'`），并用美元符号（`$`）包围数学文本，如 TeX。常规文本和数学文本可以在同一个字符串内交错。Mathtext 可以使用 Computer Modern 字体（来自 (La)TeX），STIX 字体（为与 Times 混合使用而设计）或你提供的 Unicode 字体。可以使用自定义变量 `mathtext.fontset` 选择 mathtext 字体（请参阅自定义 matplotlib）

> 注意
>
> 在Python的『narrow』构建中，如果使用 STIX 字体，你还应该将 `ps.fonttype` 和 `pdf.fonttype` 设置为 3（默认值），而不是 42。否则一些字符将不可见。

下面是个简单的例子：

```
# plain text
plt.title('alpha > beta')
```

生成 `alpha > beta`。

但是这个：

```
# math text
plt.title(r'$\alpha > \beta$')
```

生成 $\alpha > \beta$。

> 注意
>
> Mathtext 应该放在一对美元符号（`$`）之间。 为了易于显示货币值，例如 `$ 100.00`，如果整个字符串中存在单个美元符号，则它将被逐字显示为美元符号。 这是常规 TeX 的一个小改变，其中非数学文本中的美元符号必须被转义（`'$'`）。
>
> 注意
>
> 虽然一对美元符号（`$`）内的语法是 TeX 风格的，但是外面的文本不是。 特别是，字符：
>
> ```
> # $ % & ~ _ ^ \ { } \( \) \[ \]
> ```
>
> 在 TeX 中的数学模式之外有特殊的意义。 因此，根据 `rcParam text.usetex` 标志这些字符的表现有所不同。 更多信息请参阅 `usetex` 教程。

下标和上标

为了制作下标和上标，使用 `_` 或者 `^` 符号：

```
r'$\alpha_i > \beta_i$'
```

$\alpha_i > \beta_i$

一些符号会自动将它们的下标或上标放在操作符的底部或顶部，例如，为了编写 0 到无穷的 x_i 的和，你可以：

```
r'$\sum_{i=0}^\infty x_i$'
```

$$\sum_{i=0}^\infty x_i$$

分数、二项式和堆叠数

可以使用 `\frac{}{}`，`\binomial{}{}` 和 `\stackrel{}{}` 命令分别创建分数，二项式和堆叠数字：

```
r'$\frac{3}{4} \binom{3}{4} \stackrel{3}{4}$'
```

产生

$\frac{3}{4} \binom{3}{4} \genfrac{}{}{0}{}{3}{4}$

分数可以任意嵌套：

```
r'$\frac{5 - \frac{1}{x}}{4}$'
```

产生

$\frac{5 - \frac{1}{x}}{4}$

请注意，在分数周围放置圆括号和花括号需要特别注意。 这种明显的方式会产生太小的括号：

```
r'$(\frac{5 - \frac{1}{x}}{4})$'
```

$(\frac{5 - \frac{1}{x}}{4})$

解决方案是在括号前面加上 `\left` 和 `\right` 以通知解析器这些括号包含整个对象：

```
r'$\left(\frac{5 - \frac{1}{x}}{4}\right)$'
```

$\left(\frac{5 - \frac{1}{x}}{4}\right)$

根式

根式可以有 `\sqrt[]{}` 产生，例如：

```
r'$\sqrt{2}$'
```

$\sqrt{2}$

方括号内可以（可选地）设置任何底数。 请注意，底数必须是一个简单的表达式，并且不能包含布局命令，如分数或上下标：

```
r'$\sqrt[3]{x}$'
```

$\sqrt[3]{x}$

字体

用于数学符号的默认字体是斜体。

> 注意
>
> 此默认值可以使用 `mathtext.default` `rcParam` 更改。 这是非常有用的，例如，通过将其设置为 `regular` ，使用与常规非数学文本相同的字体作为数学文本。

为了修改字体，例如，以罗马字体编写 `sin` ，使用字体命令来闭合文本：

```
r'$s(t) = \mathcal{A}\mathrm{sin}(2 \omega t)$'
```

$s(t) = \mathcal{A}\mathrm{sin}(2 \omega t)$

这里 `s` 和 `t` 是斜体（默认）的变量， `sin` 是罗马字体，振幅 `A` 是书法字体。 注意在上面的例子中， `A` 和 `sin` 之间的间距被挤压。 你可以使用间距命令在它们之间添加一些空格：

```
s(t) = \mathcal{A}\/\sin(2 \omega t)
```

$s(t) = \mathcal{A}\/\sin(2 \omega t)$

所有字体的可用选项为：

命令	结果
`\mathrm{Roman}`	Roman
`\mathit{Italic}`	Italic
`\mathtt{Typewriter}`	$\mathtt{Typewriter}$
`\mathcal{CALLIGRAPHY}`	$\mathcal{CALLIGRAPHY}$

使用 STIX 字体时，你也可以选择：

命令	结果
`\mathbb{blackboard}`	$\mathbb{blackboard}$
`\mathrm{\mathbb{blackboard}}`	blackboard
`\mathfrak{Fraktur}`	$\mathfrak{Fraktur}$
`\mathsf{sansserif}`	*sansserif*
`\mathrm{\mathsf{sansserif}}`	sansserif
`\mathcircled{circled}`	ⓒⓘⓡⓒⓛⓔⓓ

还有三个全局『字体集』可供选择，它们使用 `matplotlibrc` 中的 `mathtext.fontset` 参数进行选择。

`cm` : Computer Modern (TeX)

$$\mathcal{R}\prod_{i=\alpha_{i+1}}^{\infty} a_i \sin(2\pi f x_i)$$

`stix` : STIX （为和 Times 混合使用而设计）

$$\mathcal{R}\prod_{i=\alpha_{i+1}}^{\infty} a_i \sin(2\pi f x_i)$$

`stixsans` : STIX sans-serif

$$\mathcal{R}\prod_{i=\alpha_{i+1}}^{\infty} a_i \sin(2\pi f x_i)$$

此外，你可以使用 `\mathdefault{...}` 或其别名 `\mathregular{...}` 来使用用于 mathtext 之外的常规文本的字体。 这种方法有一些限制，最明显的是，可以使用很少的符号，但可用于将数学表达式与图中的其他文本混合。

自定义字体

mathtext 还提供了一种对数学公式使用自定义字体的方法。 这种方法使用起来相当棘手，应该看做为有耐心的用户准备的试验特性。 通过将 `rcParam mathtext.fontset` 设置为 `custom` ，你可以设置以下参数，这些参数控制用于特定数学字符集的字体文件。

参数	相当于
`mathtext.it`	`\mathit{}` 默认斜体
`mathtext.rm`	`\mathrm{}` 罗马字体（upright）
`mathtext.tt`	`\mathtt{}` 打字机（monospace）
`mathtext.bf`	`\mathbf{}` 粗体
`mathtext.cal`	`\mathcal{}` 书法
`mathtext.sf`	`\mathsf{}` sans-serif

每个参数应该设置为 `fontconfig` 字体描述符（在尚未编写的字体章节中定义）。

所使用的字体应该具有 Unicode 映射，以便找到任何非拉丁字符，例如希腊语。如果要使用未包含在自定义字体中的数学符号，可以将 `rcParam mathtext.fallback_to_cm` 设置为 `True`，这将导致自定义字体中找不到特定字符时，数学文本系统使用默认的 Computer Modern 字体中的字符。

请注意，Unicode 中规定的数学字形随时间而演进，许多字体的字形对于 mathtext 可能不在正确位置。

重音符号

重音命令可以位于任何符号之前，在其上添加重音。 他们中的一些些拥有较长和较短的形式。

命令	结果
`\acute a` 或 `\'a`	$\acute{a}$
`\bar a`	$\bar{a}$
`\breve a`	$\breve{a}$
`\ddot a` 或 `\"a`	$\ddot{a}$
`\dot a` 或 `\.a`	$\dot{a}$
`\grave a` 或 `\ a``	$\grave{a}$
`\hat a` 或 `\^a`	$\hat{a}$
`\tilde a` 或 `\~a`	$\tilde{a}$
`\vec a`	$\vec{a}$
`\overline{abc}`	$\overline{abc}$

另外有两个特殊的重音符号，可以自动调整为符号的宽度：

命令	结果
`\widehat{xyz}`	$\widehat{xyz}$
`\widetilde{xyz}`	$\widetilde{xyz}$

当把重音放在小写的 `i` 和 `j` 上时应该小心。注意下面的 `\imath` 用来避免 `i` 上额外的点：

```
r"$\hat i\ \ \hat \imath$"
```

$\hat i\ \ \hat \imath$

符号

你也可以使用更大量的 TeX 符号，比如 `\infty`，`\leftarrow`，`\sum`，`\int`。

小写希腊字母				
α `\alpha`	β `\beta`	χ `\chi`	δ `\delta`	$\digamma$ `\digamma`
ϵ `\epsilon`	η `\eta`	γ `\gamma`	ι `\iota`	κ `\kappa`
λ `\lambda`	μ `\mu`	ν `\nu`	ω `\omega`	ϕ `\phi`
π `\pi`	ψ `\psi`	ρ `\rho`	σ `\sigma`	τ `\tau`
θ `\theta`	υ `\upsilon`	ε `\varepsilon`	$\varkappa$ `\varkappa`	φ `\varphi`
ϖ `\varpi`	ϱ `\varrho`	ς `\varsigma`	ϑ `\vartheta`	ξ `\xi`
ζ `\zeta`				

大写希腊字母					
Δ `\Delta`	Γ `\Gamma`	Λ `\Lambda`	Ω `\Omega`	Φ `\Phi`	Π `\Pi`
Ψ `\Psi`	Σ `\Sigma`	Θ `\Theta`	Υ `\Upsilon`	Ξ `\Xi`	$\mho$ `\mho`
∇ `\nabla`					

希伯来文			
$\aleph$ `\aleph`	$\beth$ `\beth`	$\daleth$ `\daleth`	$\gimel$ `\gimel`

分隔符				
$/$ `/`	$[$ `[`	$\Downarrow$ `\Downarrow`	$\Uparrow$ `\Uparrow`	$\Vert$ `\Vert`
$\downarrow$ `\downarrow`	$\langle$ `\langle`	$\lceil$ `\lceil`	$\lfloor$ `\lfloor`	$\llcorner$ `\llcorner`
$\rangle$ `\rangle`	$\rceil$ `\rceil`	$\rfloor$ `\rfloor`	$\ulcorner$ `\ulcorner`	$\uparrow$ `\uparrow`
$\vert$ `\vert`	$\{$ `\{`	$\Vert$ `\	`	$\}$ `\}`

大型符号				
$\bigcap$ `\bigcap`	$\bigcup$ `\bigcup`	$\bigodot$ `\bigodot`	$\bigoplus$ `\bigoplus`	$\bigotimes$ `\bigotimes`
$\biguplus$ `\biguplus`	$\bigvee$ `\bigvee`	$\bigwedge$ `\bigwedge`	$\coprod$ `\coprod`	$\int$ `\int`
$\oint$ `\oint`	$\prod$ `\prod`	$\sum$ `\sum`		

标准函数名称			
$\Pr$ `\Pr`	$\arccos$ `\arccos`	$\arcsin$ `\arcsin`	$\arctan$ `\arctan`
$\arg$ `\arg`	$\cos$ `\cos`	$\cosh$ `\cosh`	$\cot$ `\cot`
$\coth$ `\coth`	$\csc$ `\csc`	$\deg$ `\deg`	$\det$ `\det`
$\dim$ `\dim`	$\exp$ `\exp`	$\gcd$ `\gcd`	$\hom$ `\hom`
$\inf$ `\inf`	$\ker$ `\ker`	$\lg$ `\lg`	$\lim$ `\lim`
$\liminf$ `\liminf`	$\limsup$ `\limsup`	$\ln$ `\ln`	$\log$ `\log`
$\max$ `\max`	$\min$ `\min`	$\sec$ `\sec`	$\sin$ `\sin`
$\sinh$ `\sinh`	$\sup$ `\sup`	$\tan$ `\tan`	$\tanh$ `\tanh`

二元运算符和关系符号		
$\Bumpeq$ `\Bumpeq`	$\Cap$ `\Cap`	$\Cup$ `\Cup`
$\doteqdot$ `\Doteq`	$\Join$ `\Join`	$\Subset$ `\Subset`
$\Supset$ `\Supset`	$\Vdash$ `\Vdash`	$\Vvdash$ `\Vvdash`
$\approx$ `\approx`	$\approxeq$ `\approxeq`	$\ast$ `\ast`
$\asymp$ `\asymp`	$\backepsilon$ `\backepsilon`	$\backsim$ `\backsim`
$\backsimeq$ `\backsimeq`	$\barwedge$ `\barwedge`	$\because$ `\because`
$\between$ `\between`	$\bigcirc$ `\bigcirc`	$\bigtriangledown$ `\bigtriangledown`
$\bigtriangleup$ `\bigtriangleup`	$\blacktriangleleft$ `\blacktriangleleft`	$\blacktriangleright$ `\blacktriangleright`
$\bot$ `\bot`	$\bowtie$ `\bowtie`	$\boxdot$ `\boxdot`
$\boxminus$ `\boxminus`	$\boxplus$ `\boxplus`	$\boxtimes$ `\boxtimes`
$\bullet$ `\bullet`	$\bumpeq$ `\bumpeq`	$\cap$ `\cap`
$\cdot$ `\cdot`	$\circ$ `\circ`	$\circeq$ `\circeq`
$\coloneqq$ `\coloneq`	$\cong$ `\cong`	$\cup$ `\cup`
$\curlyeqprec$ `\curlyeqprec`	$\curlyeqsucc$ `\curlyeqsucc`	$\curlyvee$ `\curlyvee`
$\curlywedge$ `\curlywedge`	$\dag$ `\dag`	$\dashv$ `\dashv`

$\ddag$ `\ddag`	$\diamond$ `\diamond`	$\div$ `\div`
$\divideontimes$ `\divideontimes`	$\doteq$ `\doteq`	$\doteqdot$ `\doteqdot`
$\dotplus$ `\dotplus`	$\doublebarwedge$ `\doublebarwedge`	$\eqcirc$ `\eqcirc`
$\eqcolon$ `\eqcolon`	$\eqsim$ `\eqsim`	$\eqslantgtr$ `\eqslantgtr`
$\eqslantless$ `\eqslantless`	$\equiv$ `\equiv`	$\fallingdotseq$ `\fallingdotseq`
$\frown$ `\frown`	$\geq$ `\geq`	$\geqq$ `\geqq`
$\geqslant$ `\geqslant`	$\gg$ `\gg`	$\ggg$ `\ggg`
$\gnapprox$ `\gnapprox`	$\gneqq$ `\gneqq`	$\gnsim$ `\gnsim`
$\gtrapprox$ `\gtrapprox`	$\gtrdot$ `\gtrdot`	$\gtreqless$ `\gtreqless`
$\gtreqqless$ `\gtreqqless`	$\gtrless$ `\gtrless`	$\gtrsim$ `\gtrsim`
$\in$ `\in`	$\intercal$ `\intercal`	$\leftthreetimes$ `\leftthreetimes`
$\leq$ `\leq`	$\leqq$ `\leqq`	$\leqslant$ `\leqslant`
$\lessapprox$ `\lessapprox`	$\lessdot$ `\lessdot`	$\lesseqgtr$ `\lesseqgtr`
$\lesseqqgtr$ `\lesseqqgtr`	$\lessgtr$ `\lessgtr`	$\lesssim$ `\lesssim`
$\ll$ `\ll`	$\lll$ `\lll`	$\lnapprox$ `\lnapprox`
$\lneqq$ `\lneqq`	$\lnsim$ `\lnsim`	$\ltimes$ `\ltimes`
$\mid$ `\mid`	$\models$ `\models`	$\mp$ `\mp`
$\nVDash$ `\nVDash`	$\nVdash$ `\nVdash`	$\napprox$ `\napprox`
$\ncong$ `\ncong`	$\ne$ `\ne`	$\neq$ `\neq`
$\neq$ `\neq`	$\nequiv$ `\nequiv`	$\ngeq$ `\ngeq`
$\ngtr$ `\ngtr`	$\ni$ `\ni`	$\nleq$ `\nleq`
$\nless$ `\nless`	$\nmid$ `\nmid`	$\notin$ `\notin`
$\nparallel$ `\nparallel`	$\nprec$ `\nprec`	$\nsim$ `\nsim`
$\nsubset$ `\nsubset`	$\nsubseteq$ `\nsubseteq`	$\nsucc$ `\nsucc`
$\nsupset$ `\nsupset`	$\nsupseteq$ `\nsupseteq`	$\ntriangleleft$ `\ntriangleleft`

$\ntrianglelefteq$ `\ntrianglelefteq`	$\ntriangleright$ `\ntriangleright`	$\ntrianglerighteq$ `\ntrianglerighteq`
$\nvDash$ `\nvDash`	$\nvdash$ `\nvdash`	$\odot$ `\odot`
$\ominus$ `\ominus`	$\oplus$ `\oplus`	$\oslash$ `\oslash`
$\otimes$ `\otimes`	$\parallel$ `\parallel`	$\perp$ `\perp`
$\pitchfork$ `\pitchfork`	$\pm$ `\pm`	$\prec$ `\prec`
$\precapprox$ `\precapprox`	$\preccurlyeq$ `\preccurlyeq`	$\preceq$ `\preceq`
$\precnapprox$ `\precnapprox`	$\precnsim$ `\precnsim`	$\precsim$ `\precsim`
$\propto$ `\propto`	$\rightthreetimes$ `\rightthreetimes`	$\risingdotseq$ `\risingdotseq`
$\rtimes$ `\rtimes`	$\sim$ `\sim`	$\simeq$ `\simeq`
$\slash$ `\slash`	$\smile$ `\smile`	$\sqcap$ `\sqcap`
$\sqcup$ `\sqcup`	$\sqsubset$ `\sqsubset`	$\sqsubset$ `\sqsubset`
$\sqsubseteq$ `\sqsubseteq`	$\sqsupset$ `\sqsupset`	$\sqsupset$ `\sqsupset`
$\sqsupseteq$ `\sqsupseteq`	$\star$ `\star`	$\subset$ `\subset`
$\subseteq$ `\subseteq`	$\subseteqq$ `\subseteqq`	$\subsetneq$ `\subsetneq`
$\subsetneqq$ `\subsetneqq`	$\succ$ `\succ`	$\succapprox$ `\succapprox`
$\succcurlyeq$ `\succcurlyeq`	$\succeq$ `\succeq`	$\succnapprox$ `\succnapprox`
$\succnsim$ `\succnsim`	$\succsim$ `\succsim`	$\supset$ `\supset`
$\supseteq$ `\supseteq`	$\supseteqq$ `\supseteqq`	$\supsetneq$ `\supsetneq`
$\supsetneqq$ `\supsetneqq`	$\therefore$ `\therefore`	$\times$ `\times`
$\top$ `\top`	$\triangleleft$ `\triangleleft`	$\trianglelefteq$ `\trianglelefteq`
$\triangleq$ `\triangleq`	$\triangleright$ `\triangleright`	$\trianglerighteq$ `\trianglerighteq`
$\uplus$ `\uplus`	$\vDash$ `\vDash`	$\varpropto$ `\varpropto`
$\vartriangleleft$ `\vartriangleleft`	$\vartriangleright$ `\vartriangleright`	$\vdash$ `\vdash`
$\vee$ `\vee`	$\veebar$ `\veebar`	$\wedge$ `\wedge`
$\wr$ `\wr`		

箭头符号		

$\Downarrow$ `\Downarrow`	$\Leftarrow$ `\Leftarrow`
$\Leftrightarrow$ `\Leftrightarrow`	$\Lleftarrow$ `\Lleftarrow`
$\Longleftarrow$ `\Longleftarrow`	$\Longleftrightarrow$ `\Longleftrightarrow`
$\Longrightarrow$ `\Longrightarrow`	$\Lsh$ `\Lsh`
$\Nearrow$ `\Nearrow`	$\Nwarrow$ `\Nwarrow`
$\Rightarrow$ `\Rightarrow`	$\Rrightarrow$ `\Rrightarrow`
$\Rsh$ `\Rsh`	$\Searrow$ `\Searrow`
$\Swarrow$ `\Swarrow`	$\Uparrow$ `\Uparrow`
$\Updownarrow$ `\Updownarrow`	$\circlearrowleft$ `\circlearrowleft`
$\circlearrowright$ `\circlearrowright`	$\curvearrowleft$ `\curvearrowleft`
$\curvearrowright$ `\curvearrowright`	$\dashleftarrow$ `\dashleftarrow`
$\dashrightarrow$ `\dashrightarrow`	$\downarrow$ `\downarrow`
$\downdownarrows$ `\downdownarrows`	$\downharpoonleft$ `\downharpoonleft`
$\downharpoonright$ `\downharpoonright`	$\hookleftarrow$ `\hookleftarrow`
$\hookrightarrow$ `\hookrightarrow`	$\leadsto$ `\leadsto`
$\leftarrow$ `\leftarrow`	$\leftarrowtail$ `\leftarrowtail`
$\leftharpoondown$ `\leftharpoondown`	$\leftharpoonup$ `\leftharpoonup`
$\leftleftarrows$ `\leftleftarrows`	$\leftrightarrow$ `\leftrightarrow`
$\leftrightarrows$ `\leftrightarrows`	$\leftrightharpoons$ `\leftrightharpoons`
$\leftrightsquigarrow$ `\leftrightsquigarrow`	$\leftsquigarrow$ `\leftsquigarrow`
$\longleftarrow$ `\longleftarrow`	$\longleftrightarrow$ `\longleftrightarrow`
$\longmapsto$ `\longmapsto`	$\longrightarrow$ `\longrightarrow`
$\looparrowleft$ `\looparrowleft`	$\looparrowright$ `\looparrowright`
$\mapsto$ `\mapsto`	$\multimap$ `\multimap`
$\nLeftarrow$ `\nLeftarrow`	$\nLeftrightarrow$ `\nLeftrightarrow`
$\nRightarrow$ `\nRightarrow`	$\nearrow$ `\nearrow`
$\nleftarrow$ `\nleftarrow`	$\nleftrightarrow$ `\nleftrightarrow`

$\nrightarrow$	\nrightarrow	$\nwarrow$	\nwarrow
$\rightarrow$	\rightarrow	$\rightarrowtail$	\rightarrowtail
$\rightharpoondown$	\rightharpoondown	$\rightharpoonup$	\rightharpoonup
$\rightleftarrows$	\rightleftarrows	$\rightleftarrows$	\rightleftarrows
$\rightleftharpoons$	\rightleftharpoons	$\rightleftharpoons$	\rightleftharpoons
$\rightrightarrows$	\rightrightarrows	$\rightrightarrows$	\rightrightarrows
$\rightsquigarrow$	\rightsquigarrow	$\searrow$	\searrow
$\swarrow$	\swarrow	$\to$	\to
$\twoheadleftarrow$	\twoheadleftarrow	$\twoheadrightarrow$	\twoheadrightarrow
$\uparrow$	\uparrow	$\updownarrow$	\updownarrow
$\updownarrow$	\updownarrow	$\upharpoonleft$	\upharpoonleft
$\upharpoonright$	\upharpoonright	$\upuparrows$	\upuparrows

杂项符号		
$ `\$`	Å `\AA`	Ⅎ `\Finv`
⅁ `\Game`	ℑ `\Im`	¶ `\P`
ℜ `\Re`	§ `\S`	∠ `\angle`
‵ `\backprime`	★ `\bigstar`	■ `\blacksquare`
▲ `\blacktriangle`	▼ `\blacktriangledown`	⋯ `\cdots`
✓ `\checkmark`	® `\circledR`	Ⓢ `\circledS`
♣ `\clubsuit`	∁ `\complement`	© `\copyright`
⋱ `\ddots`	♢ `\diamondsuit`	ℓ `\ell`
∅ `\emptyset`	ð `\eth`	∃ `\exists`
♭ `\flat`	∀ `\forall`	ħ `\hbar`
♡ `\heartsuit`	ℏ `\hslash`	∭ `\iiint`
∬ `\iint`	∬ `\iint`	ı `\imath`
∞ `\infty`	ȷ `\jmath`	… `\ldots`
∡ `\measuredangle`	♮ `\natural`	¬ `\neg`
∄ `\nexists`	∰ `\oiiint`	∂ `\partial`
′ `\prime`	♯ `\sharp`	♠ `\spadesuit`
∢ `\sphericalangle`	ß `\ss`	▽ `\triangledown`
∅ `\varnothing`	△ `\vartriangle`	⋮ `\vdots`
℘ `\wp`	¥ `\yen`	

如果特定符号没有名称（对于 STIX 字体中的许多较为模糊的符号也是如此），也可以使用 Unicode 字符：

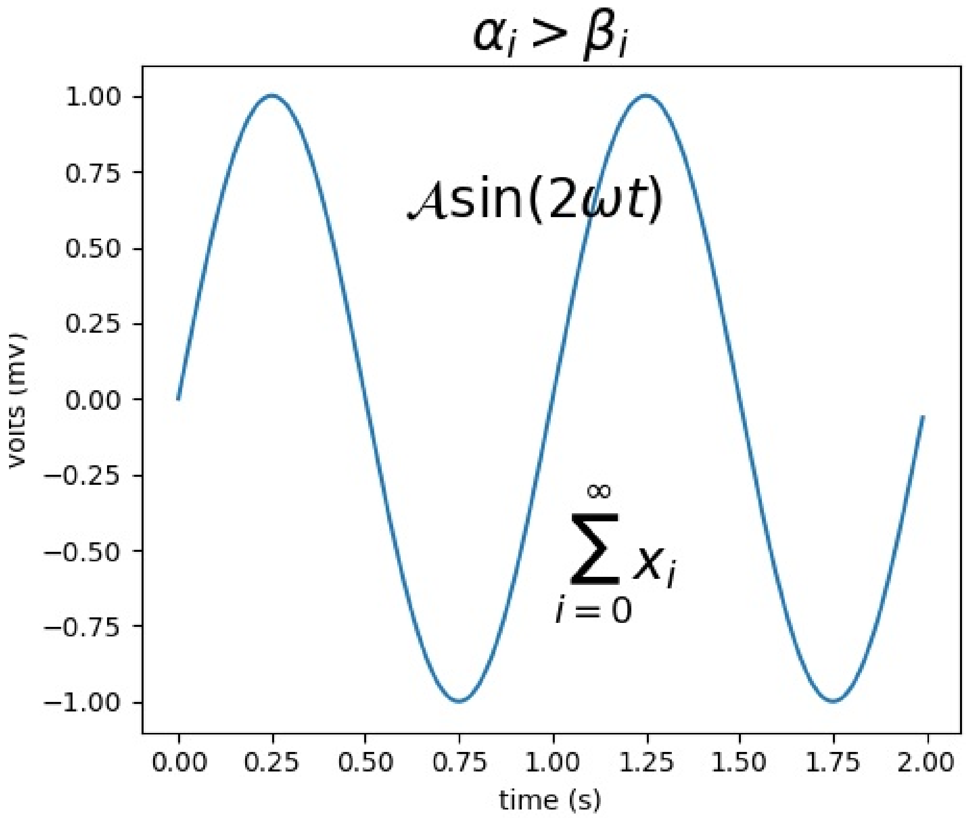
```
ur'$\u23ce$'
```

示例

下面是个示例，在上下文中展示了许多这些特性。

```
import numpy as np
import matplotlib.pyplot as plt
t = np.arange(0.0, 2.0, 0.01)
s = np.sin(2*np.pi*t)

plt.plot(t,s)
plt.title(r'$\alpha_i > \beta_i$', fontsize=20)
plt.text(1, -0.6, r'$\sum_{i=0}^\infty x_i$', fontsize=20)
plt.text(0.6, 0.6, r'$\mathcal{A}\mathrm{sin}(2 \omega t)$',
         fontsize=20)
plt.xlabel('time (s)')
plt.ylabel('volts (mV)')
plt.show()
```



使用 LaTeX 渲染文本

> 原文：Text rendering With LaTeX
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

Matplotlib 可以选择使用 LaTeX 来管理所有文本布局。 此选项可用于以下后端：

- Agg
- PS
- PDF

LaTeX 选项通过在 `rc` 设置中设置 `text.usetex:True` 来激活。 使用 matplotlib 的 LaTeX 支持的文本处理会慢于 matplotlib 的非常强大的 mathtext，但是更灵活，因为可以使用不同的 LaTeX 包（字体包，数学包等）。 结果会十分惊人，特别是当你在图形中使用和主文档相同的字体。

Matplotlib 的 LaTeX 支持需要可用的 LaTeX 安装版本，dvipng（可能包括在你的 LaTeX 安装中）和 Ghostscript（建议使用 GPL Ghostscript 8.60 或更高版本）。 这些外部依赖的可执行文件必须都位于你的 `PATH` 中。

有几个选项需要提及，可以使用 `rc` 设置更改它们。 这里是一个 `matplotlibrc` 示例文件：

```
font.family         : serif
font.serif          : Times, Palatino, New Century Schoolbook, Bookman, Computer Modern Roman
font.sans-serif     : Helvetica, Avant Garde, Computer Modern Sans serif
font.cursive        : Zapf Chancery
font.monospace      : Courier, Computer Modern Typewriter

text.usetex         : true
```

每个系列中的第一个有效字体是要加载的字体。 如果未指定字体，则默认使用 Computer Modern 字体。 所有其他字体是 Adobe 字体。 Times 和 Palatino 每个都有自己附带的数学字体，而其他 Adobe 衬线字体使用 Computer Modern 数学字体。 有关更多详细信息，请参阅 PSNFSS 文档。

要使用 LaTeX 并选择 Helvetica 作为默认字体，但不编辑 `matplotlibrc`，使用：

```
from matplotlib import rc
rc('font',**{'family':'sans-serif','sans-serif':['Helvetica']})
## for Palatino and other serif fonts use:
#rc('font',**{'family':'serif','serif':['Palatino']})
rc('text', usetex=True)
```

这里是标准的示例，`tex_demo.py`：

```
"""
Demo of TeX rendering.

You can use TeX to render all of your matplotlib text if the rc
parameter text.usetex is set.  This works currently on the agg a
nd ps
backends, and requires that you have tex and the other dependenc
ies
described at http://matplotlib.org/users/usetex.html
properly installed on your system.  The first time you run a scr
ipt
you will see a lot of output from tex and associated tools.  The
 next
time, the run may be silent, as a lot of the information is cach
ed in
~/.tex.cache

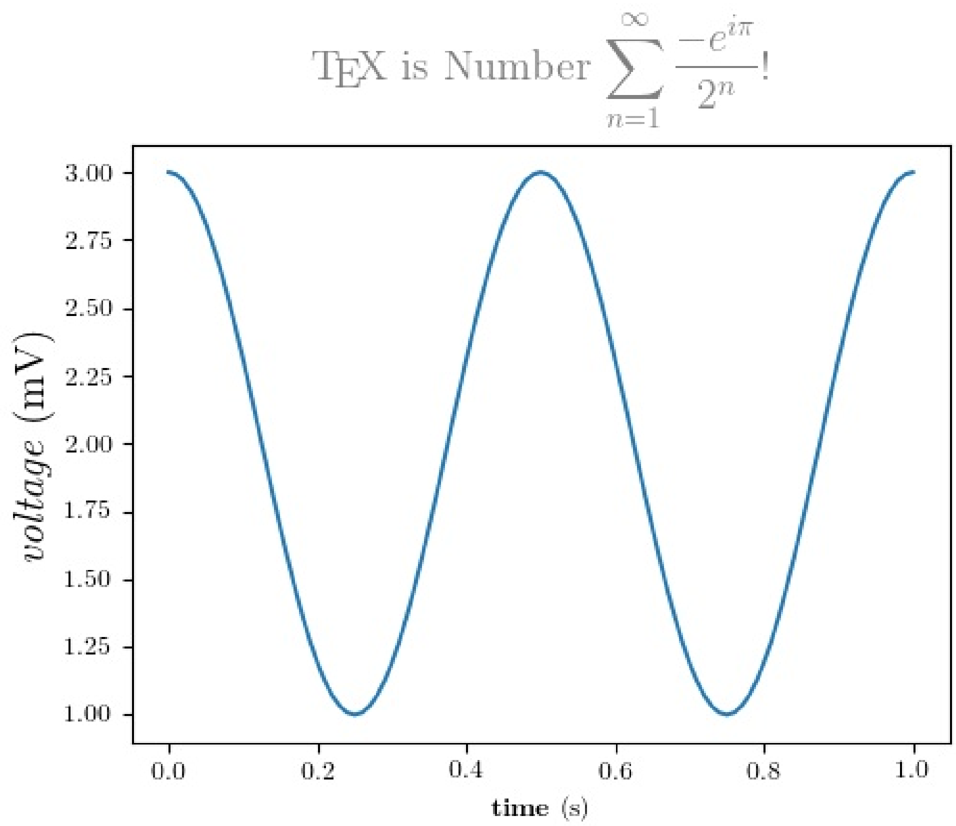
"""
import numpy as np
import matplotlib.pyplot as plt


# Example data
t = np.arange(0.0, 1.0 + 0.01, 0.01)
s = np.cos(4 * np.pi * t) + 2

plt.rc('text', usetex=True)
plt.rc('font', family='serif')
plt.plot(t, s)

plt.xlabel(r'\textbf{time} (s)')
plt.ylabel(r'\textit{voltage} (mV)',fontsize=16)
plt.title(r"\TeX\ is Number "
          r"$\displaystyle\sum_{n=1}^\infty\frac{-e^{i\pi}}{2^n}
$!",
          fontsize=16, color='gray')
# Make room for the ridiculously large title.
plt.subplots_adjust(top=0.8)

plt.savefig('tex_demo')
plt.show()
```



要注意数学显示模式（`$$ e=mc^2 $$`）是不支持的，但是添加命令`\displaystyle`之后会产生相同结果，就像`tex_demo.py`中那样。

> 注意
>
> 一些字符在 TeX 中需要特别转义，例如：
>
> ```
> # $ % & ~ _ ^ \ { } \( \) \[ \]
> ```
>
> 所以，这些字符会根据`rcParam text.usetex`标志位而表现不同。

在 TeX 中使用 Unicode

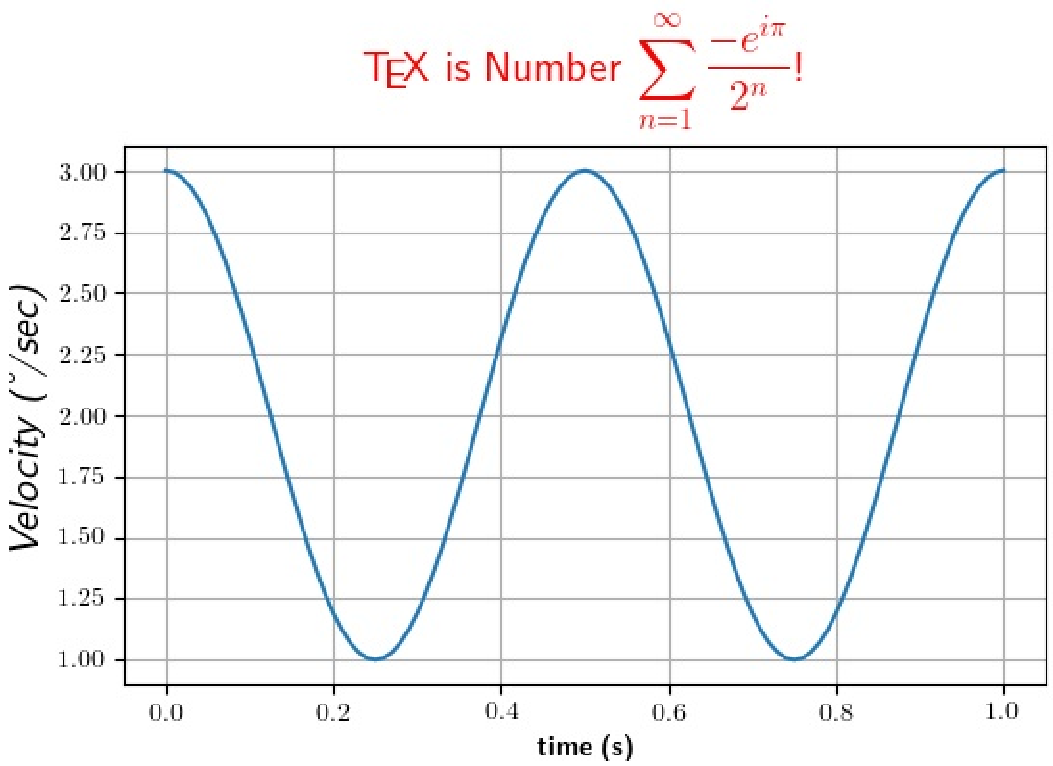
也可以在 LaTeX 文本管理器中使用 unicode 字符串，这里是从`tex_unicode_demo.py`中获取的示例：

```
# -*- coding: utf-8 -*-
"""
This demo is tex_demo.py modified to have unicode. See that file
 for
more information.
"""

from __future__ import unicode_literals
import numpy as np
import matplotlib
matplotlib.rcParams['text.usetex'] = True
matplotlib.rcParams['text.latex.unicode'] = True
import matplotlib.pyplot as plt

plt.figure(1, figsize=(6, 4))
ax = plt.axes([0.1, 0.1, 0.8, 0.7])
t = np.arange(0.0, 1.0 + 0.01, 0.01)
s = np.cos(2*2*np.pi*t) + 2
plt.plot(t, s)

plt.xlabel(r'\textbf{time (s)}')
plt.ylabel('\\textit{Velocity (\u00B0/sec)}', fontsize=16)
plt.title(r'\TeX\ is Number $\displaystyle\sum_{n=1}^\infty'
          r'\frac{-e^{i\pi}}{2^n}$!', fontsize=16, color='r')
plt.grid(True)
plt.show()
```



Postscript 选项

为了生成可以嵌入到新 LaTeX 文档中的 postscript 封装文件，matplotlib 的默认行为是提取输出，这会删除 LaTeX 使用的一些 postscript 操作符，这些操作符在 eps 文件中是非法的。 此步骤产生的结果对于一些用户可能是不可接受的，因为文本被粗略地光栅化并且被转换为位图，而不像标准 Postscript 那样是可缩放的，并且文本是不可搜索的。 一种解决方法是在你的 `rc` 设置中将 `ps.distiller.res` 设置为较高的值（可能为 6000），这将产生较大的文件，但可能看起来更好并能够合理缩放。 更好的解决方法需要 Poppler 或 Xpdf，可以通过将 `ps.usedistiller rc` 设置更改为 `xpdf` 来激活。 此替代方案产生 postscript 而不光栅化文本，因此它能够正确缩放，可以在 Adobe Illustrator 中编辑，并搜索 `pdf` 文档中的文本。

可能的问题

- 在 Windows 上，可能需要修改 `PATH` 环境变量来包含 latex，dvipng 和 ghostscript 可执行文件的目录。详细信息请参阅环境变量和在 Windows 中设置环境变量。
- 使用 MiKTeX 与 Computer Modern 字体，如果你得到奇怪的 *Agg 和 PNG 结果，访问 `MiKTeX/Options` 并更新你的格式文件。
- 字体在屏幕上看起来糟糕。你可能正在运行 Mac OS，在 mac 上的老版本 dvipng 运行着一些有趣的事情。在你的 `matplotlibrc` 文件中设置 `text.dvipnghack:True` 。
- 在 Ubuntu 和 Gentoo 上，texlive 基本安装不附带 type1cm 包。你可能需要安装一些额外的包，来获得所有与其它 LaTeX 分发版捆绑的特性。
- matplotlib 已经取得了一些进展，所以可以直接使用 `dvi` 文件进行文本布局。这允许 LaTeX 用于具有 `pdf` 和 `svg` 后端的文本布局，以及 *Agg 和 PS 后端。在将来，LaTeX 安装可能是唯一的外部依赖。

故障排除

- 尝试删除 `.matplotlib/tex.cache` 目录。 如果你不知道 `.matplotlib` 在哪里，请参见 matplotlib 配置和缓存目录位置。
- 确保 LaTeX，dvipng 和 ghostscript 都正常工作，并存在于你的 `PATH` 中。
- 确保你想要做的事情在 LaTeX 文档中可实现，你的 LaTeX 语法是有效的，并且你正在使用原始字符串，如果必要，以避免意外的转义序列。
- 邮件列表上报告的大多数问题已通过升级 Ghostscript 来清除。 如果可能的话，请尝试升级到最新版本，然后向列表报告问题。
- `text.latex.preamble rc` 设置不受官方支持。 此选项提供了大量的灵活性和导致问题的许多方法。 请在向邮件列表报告问题之前禁用此选项。
- 如果仍需要帮助，请参阅获取帮助。

XeLaTeX/LuaLaTeX 设置

> 原文：Typesetting With XeLaTeX/LuaLaTeX
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

使用 pgf 后端，matplotlib 可以将图形导出为可以使用 pdflatex，xelatex 或 lualatex 处理的 pgf 绘图命令。 XeLaTeX 和 LuaLaTeX 具有完整的 unicode 支持，可以使用安装在操作系统中的任何字体，利用 OpenType，AAT 和 Graphite 的高级排版功能。 由 `plt.savefig('figure.pgf')` 创建的 Pgf 图片可以作为原始命令嵌入到 LaTeX 文档中。 图形也可以通过切换到该后端，直接编译并使用 `plt.savefig('figure.pdf')` 保存到 PDF。

```
matplotlib.use('pgf')
```

或者为处理 PDF 输出而注册它：

```
from matplotlib.backends.backend_pgf import FigureCanvasPgf
matplotlib.backend_bases.register_backend('pdf', FigureCanvasPgf
)
```

第二种方法允许你继续使用常规的交互式后端，并从图形用户界面保存 xelatex，lualatex 或 pdflatex 编译的 PDF 文件。

Matplotlib 的 pgf 支持需要最新的 LaTeX 安装，包括 TikZ/PGF 软件包（如 TeXLive），最好安装 XeLaTeX 或 LuaLaTeX。 如果你的系统上存在 pdftocairo 或 ghostscript，也可以选择将图形保存为 PNG 图像。 所有应用程序的可执行文件必须位于 `PATH` 中。

控制 pgf 后端行为的 Rc 参数：

参数	文档
`pgf.preamble`	包含在 LaTeX 序言中的行
`pgf.rcfonts`	使用 fontspec 软件包从 rc 参数设置字体
`pgf.texsystem`	`xelatex`（默认），`lualatex` 或者 `pdflatex`

> 注意
>
> TeX 定义了一系列特殊字符，例如：
>
> ```
> # $ % & ~ _ ^ \ { }
> ```
>
> 通常，这些字符必须正确转义。一些字符（`_`，`^`，`%`）会自动在数学环境之外转义。

字体规定

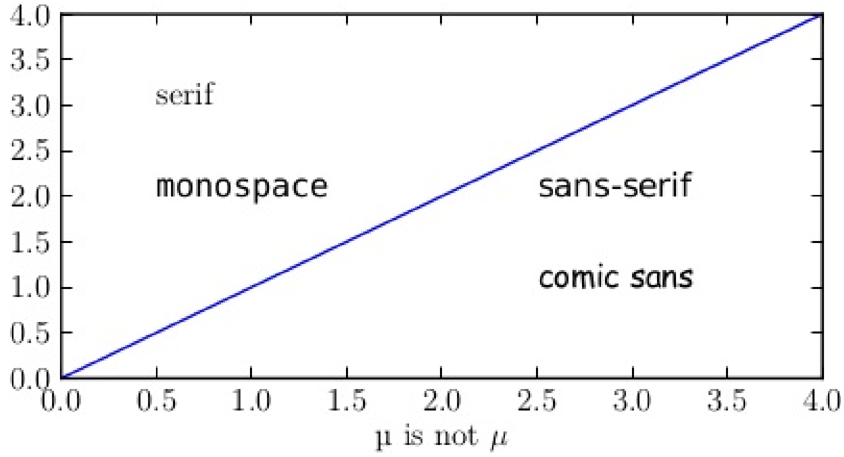
用于获取文本元素大小，或将图形编译为 PDF 的字体通常在 matplotlib rc 参数中定义。你还可以通过清除 `font.serif`，`font.sans-serif` 或 `font.monospace` 的列表来使用 LaTeX 默认的 Computer Modern 字体。请注意，这些字体的字形覆盖范围非常有限。如果要保留 Computer Modern 字体，但需要扩展 Unicode 编码支持，请考虑安装 Computer Modern Unicode 字体 CMU Serif，CMU Sans Serif 等。

保存到 `.pgf` 时，matplotlib 用于图形布局的字体配置包含在文本文件的标题中。

```
# -*- coding: utf-8 -*-

import matplotlib as mpl
mpl.use("pgf")
pgf_with_rc_fonts = {
    "font.family": "serif",
    "font.serif": [],                    # use latex default serif font
    "font.sans-serif": ["DejaVu Sans"], # use a specific sans-serif font
}
mpl.rcParams.update(pgf_with_rc_fonts)

import matplotlib.pyplot as plt
plt.figure(figsize=(4.5,2.5))
plt.plot(range(5))
plt.text(0.5, 3., "serif")
plt.text(0.5, 2., "monospace", family="monospace")
plt.text(2.5, 2., "sans-serif", family="sans-serif")
plt.text(2.5, 1., "comic sans", family="Comic Sans MS")
plt.xlabel(u"µ is not $\\mu$")
plt.tight_layout(.5)
```



自定义序言

通过将你的命令添加到序言中，你可以完全自定义它。 如果要配置数学字体（例如使用 unicode-math）或加载其他软件包，请使用 `pgf.preamble` 参数。 此外，如果你想自己做字体配置，而不是使用 rc 参数中指定的字体，请确保禁用 `pgf.rcfonts` 。

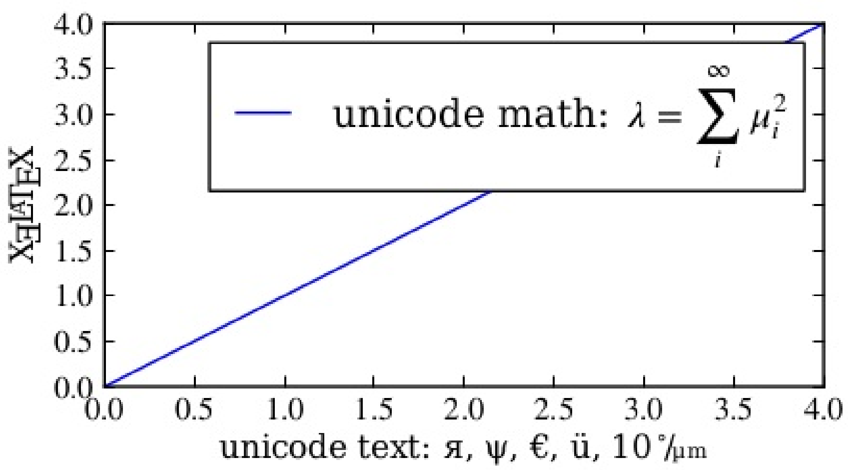
```
# -*- coding: utf-8 -*-
from __future__ import (absolute_import, division, print_functio
n,
                        unicode_literals)

import six

import matplotlib as mpl
mpl.use("pgf")
pgf_with_custom_preamble = {
    "font.family": "serif", # use serif/main font for text eleme
nts
    "text.usetex": True,    # use inline math for ticks
    "pgf.rcfonts": False,   # don't setup fonts from rc paramete
rs
    "pgf.preamble": [
         "\\usepackage{units}",         # load additional packag
es
         "\\usepackage{metalogo}",
         "\\usepackage{unicode-math}",  # unicode math setup
         r"\setmathfont{xits-math.otf}",
         r"\setmainfont{DejaVu Serif}", # serif font via preamble

         ]
}
mpl.rcParams.update(pgf_with_custom_preamble)

import matplotlib.pyplot as plt
plt.figure(figsize=(4.5,2.5))
plt.plot(range(5))
plt.xlabel("unicode text: я, ψ, €, ü, \\unitfrac[10]{°}{µm}")
plt.ylabel("\\XeLaTeX")
plt.legend(["unicode math: $λ=∑_i^∞ μ_i^2$"])
plt.tight_layout(.5)
```



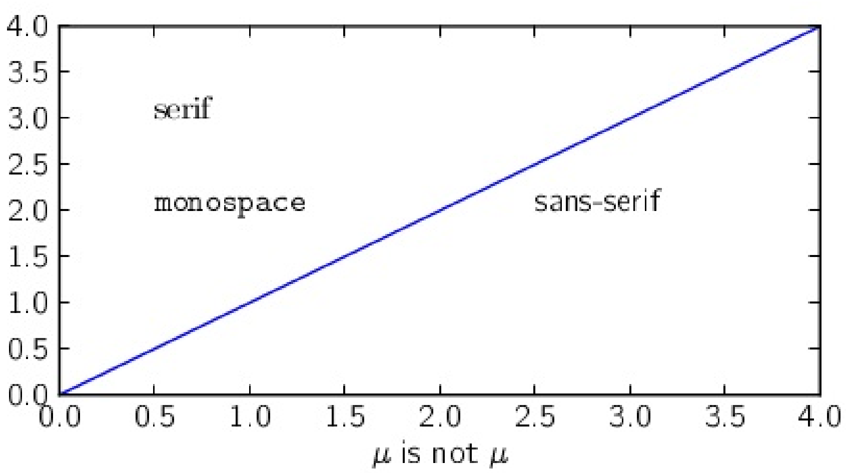
选择 TeX 系统

matplotlib 使用的 TeX 系统由 `pgf.texsystem` 参数选择。 可能的值为 `xelatex` （默认值）， `lualatex` 和 `pdflatex` 。 请注意，当选择 `pdflatex` 时，必须在序言中配置字体和 unicode 处理。

```
# -*- coding: utf-8 -*-

import matplotlib as mpl
mpl.use("pgf")
pgf_with_pdflatex = {
    "pgf.texsystem": "pdflatex",
    "pgf.preamble": [
         r"\usepackage[utf8x]{inputenc}",
         r"\usepackage[T1]{fontenc}",
         r"\usepackage{cmbright}",
         ]
}
mpl.rcParams.update(pgf_with_pdflatex)

import matplotlib.pyplot as plt
plt.figure(figsize=(4.5,2.5))
plt.plot(range(5))
plt.text(0.5, 3., "serif", family="serif")
plt.text(0.5, 2., "monospace", family="monospace")
plt.text(2.5, 2., "sans-serif", family="sans-serif")
plt.xlabel(u"µ is not $\\mu$")
plt.tight_layout(.5)
```



故障排除

请注意，在一些 Linux 发行版和 MiKTeX 安装中发现的 TeX 包已经过时了。确保更新你的软件包目录并升级或安装最新的 TeX 发行版。 在 Windows 上，可能需要修改 `PATH` 环境变量来包含 latex，dvipng 和 ghostscript 可执行文件的目录。详细信

息请参阅环境变量和在窗口中设置环境变量。 Windows 上的限制会导致后端保留由应用程序打开的文件句柄。因此，可能无法删除相应的文件，直到应用程序关闭（参见 #1324 ）。 有时保存到 png 图像的图形中的字体非常糟糕。这在 pdftocairo 工具不可用，并且 ghostscript 用于 pdf 到 png 的转换时发生。 确保你想要做的事情在 LaTeX 文档中可实现，你的 LaTeX 语法是有效的，并且你正在使用原始字符串，如果必要的话，避免意外的转义序列。 `pgf.preamble rc` 设置提供了大量的灵活性，以及导致问题的许多方法。遇到问题时，尝试最小化或禁用自定义序言。 配置 unicode-math 环境可能有点棘手。例如 TeXLive 分发版提供了一组通常不在系统范围内安装的数学字体。与 LuaLatex 不同的是，XeTeX 不能根据名字找到这些字体，这就是你可能必须指定 `\setmathfont{xits-math.otf}` ，而不是 `\setmathfont{XITS Math}` 的原因，或者使字体可用于你的操作系统。更多详细信息请参阅这个 `tex.stackexchange.com` 的问题。 如果 matplotlib 使用的字体配置不同于你的 LaTeX 文档中的字体设置，则导入图形中的文本元素对齐可能会关闭。如果你不确定 matplotlib 用于布局的字体，请检查 `.pgf` 文件的标题。 如果图中有很多对象，矢量图像和 `.pgf` 文件可能变得臃肿。这可能是图像处理或非常大的散点图的情况。在极端情况下，这可能导致 TeX 内存不足： `TeX capacity exceeded, sorry` （TeX 容量过大，对不起）。你可以配置 LaTeX 来增加可用于生成 `.pdf` 图像的内存量，请见 `tex.stackexchange.com` 上讨论的问题。另一种方法是使用 `rasterized = True` 关键字，或者根据本示例的 `.set_rasterized(True)` 『栅格化』图形的某些导致问题部分。 如果你仍需要帮助，请参阅获取帮助。

颜色

指定颜色

> 原文：Specifying Colors
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

在 matplotlib 的几乎所有地方，用户都可以指定颜色，它可以以如下形式提供：

- RGB 或者 RGBA 浮点值元组，`[0, 1]` 之间，例如 `(0.1, 0.2, 0.5)` 或者 `(0.1, 0.2, 0.5, 0.3)` 。
- RGB 或者 RGBA 十六进制字符串，例如 `#0F0F0F` 或者 `#0F0F0F0F` 。
- `[0, 1]` 之间的浮点值的字符串表示，用于表示灰度，例如 `0.5` 。
- `{'b', 'g', 'r', 'c', 'm', 'y', 'k', 'w'}` 之一。
- X11/CSS4 颜色名称。
- XKCD 颜色之一，以 `'xkcd:'` 为前缀，例如 `'xkcd:sky blue'` 。
- `{'C0', 'C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7', 'C8', 'C9'}` 之一。
- `{'tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple',` 之一。这是 T10 调色板的 Tableau 颜色（默认的色相环）。

所有颜色字符串都是大小写敏感的。

`CN` 颜色选择

颜色可以通过匹配正则表达式 `C[0-9]` 的字符串来指定。 这可以在任何当前接受颜色的地方传递，并且可以在 `matplotlib.Axes.plot` 的 `format-string` 中用作“单个字符颜色”。

单个数字是默认属性环的索引（ `matplotlib.rcParams['axes.prop_cycle']` ）。 如果属性环不包括 `'color'` ，则返回黑色。 在创建艺术家时会对颜色求值。 例如：

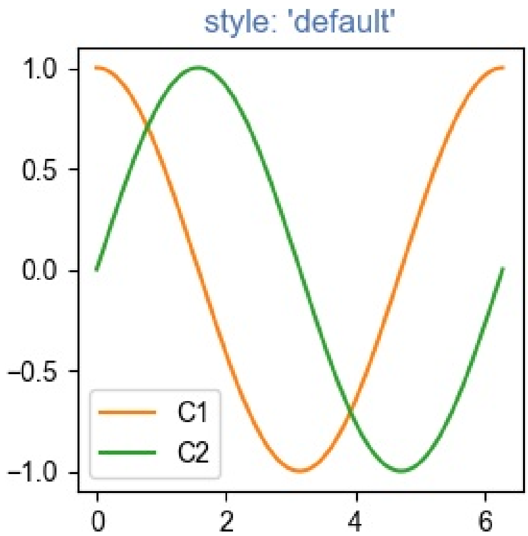
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
th = np.linspace(0, 2*np.pi, 128)

def demo(sty):
    mpl.style.use(sty)
    fig, ax = plt.subplots(figsize=(3, 3))

    ax.set_title('style: {!r}'.format(sty), color='C0')

    ax.plot(th, np.cos(th), 'C1', label='C1')
    ax.plot(th, np.sin(th), 'C2', label='C2')
    ax.legend()

demo('default')
```





自定义 matplotlib

> 原文：Customizing matplotlib
>
> 译者：飞龙
>
>

使用样式表自定义绘图

`style` 包为易于切换的绘图『样式』增加了支持，它们与 `matplotlibrc` 文件参数相同。

有一些预定义样式由 `matplotlib` 提供。例如，有一个名为『ggplot』的预定义样式，它模拟 `ggplot`（R 的一种流行的绘图软件包）的美学。为了使用此样式，只需添加：

```
>>> import matplotlib.pyplot as plt
>>> plt.style.use('ggplot')
```

为了列出所有可用样式，使用：

```
>>> print(plt.style.available)
```

定义你自己的样式

你可以创建自定义样式，并通过以样式表的路径或 URL 调用 `style.use` 来使用它们。或者，如果将 `<style-name> mplstyle` 文件添加到 `mpl_configdir /stylelib` 中，你可以通过调用 `style.use(<style-name>)` 重复使用自定义样式表。默认情况下 `mpl_configdir` 应该是 `~/.config/matplotlib`，但你可以使用 `matplotlib.get_configdir()` 检查你的位置，你可能需要创建这个目录。请注意，如果样式具有相同的名称，`mpl_configdir/stylelib` 中的自定义样式表将覆盖由 `matplotlib` 定义的样式表。

例如，你可能想要使用以下命令创建 `mpl_configdir/stylelib/presentation.mplstyle`：

```
axes.titlesize : 24
axes.labelsize : 20
lines.linewidth : 3
lines.markersize : 10
xtick.labelsize : 16
ytick.labelsize : 16
```

然后，当你想要将一个为纸张设计的地图迁移到演示文档中时，你可以添加：

```
>>> import matplotlib.pyplot as plt
>>> plt.style.use('presentation')
```

组合样式

样式表为组合在一起而设计。 因此，你可以拥有一个自定义颜色的样式表和一个单独的样式表，用于更改演示文档的元素大小。 这些样式可以通过传递样式列表轻松组合：

```
>>> import matplotlib.pyplot as plt
>>> plt.style.use(['dark_background', 'presentation'])
```

请注意，右侧的样式将覆盖已经由左侧样式定义的值。

临时样式

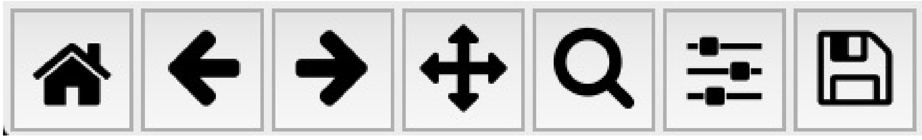
如果只想对特定的代码块使用样式，但不想更改全局样式，那么样式包提供了一个上下文管理器，用于将更改限制于特定范围。 要隔离你的样式更改，你可以编写以下内容：

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>>
>>> with plt.style.context(('dark_background')):
>>>     plt.plot(np.sin(np.linspace(0, 2 * np.pi)), 'r-o')
>>>
>>> # Some plotting code with the default style
>>>
>>> plt.show()
```

交互式绘图

交互式导航

> 原文：Interactive navigation
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0



所有图形窗口都带有导航工具栏，可用于浏览数据集。 以下是工具栏底部的每个按钮的说明：



`Home`（首页）、`Forward`（前进）和 `Back`（后退）按钮：

这些类似于 Web 浏览器的前进和后退按钮。 它们用于在之前定义的视图之间来回浏览。 它们没有意义，除非你已经使用平移和缩放按钮访问了其他地方。 这类似于尝试在访问新页面之前单击 Web 浏览器上的返回 - 什么都不会发生。 首页总是你第一个浏览的页面，以及你的数据的默认视图。 对于 `Home`，`Forward` 和 `Back`，应该将其看做 Web浏览器，其中的数据视图是网页。 使用 `Pan` 和 `Zoom` 来定义新视图。



`Pan/Zoom`（平移/缩放）按钮

此按钮有两种模式：平移和缩放。 单击工具栏按钮激活平移和缩放，然后将鼠标放在轴域的某个地方。 按住鼠标左键并将其拖动到新位置来平移图形。 当你释放它时，你按下的点处的数据将移动到你释放的点。 如果在平移时按 `'x'` 或 `'y'`，移动会分别限制在 `x` 或 `y` 轴。 按鼠标右键并将其拖动到新位置来进行缩放。 向右移动使 `x` 轴成比例放大，或者向左移动成比例缩小。 `y` 轴和上/下移动同上。 开始缩放时鼠标下的点会保持静止，你可以缩放图形中的其它任意点。 你可以使用快捷键 `'x'`，`'y'` 或 `CONTROL` 分别将缩放约束为 `x` 轴，`y` 轴或保留宽高比。

使用极坐标绘图时，平移和缩放功能的行为不同。 可以使用鼠标左键拖动半径轴标签。 可以使用鼠标右键放大和缩小半径刻度。



`Zoom-to-rectangle`（缩放到矩形）按钮

单击此工具栏按钮以激活此模式。 将鼠标放在轴域的某处，然后按鼠标左键。 在按住按钮的同时拖动鼠标到新位置并释放。 轴域会放大并限制于你定义的矩形。 在此模式中还有一个实验性的 `zoom out to rectangle`（缩小到矩形），使用右键，将整个轴域缩小并放置在矩形定义的区域中。



`Subplot-configuration` （子图配置）按钮

使用此工具配置子图的参数：左边距，右边距，上边距，下边距，行间隔和列间隔。



`Save` （保存）按钮

单击此按钮可启动文件保存对话框。 你可以使用以下扩展名保存文件： `png` ， `ps` ， `eps` ， `svg` 和 `pdf` 。

浏览快捷键

下表包含所有默认的快捷键，可以使用 `matplotlibrc` （ `#keymap.*` ）覆盖。

命令	快捷键
主页/重置	`h` 、 `r` 或 `home`
后退	`c` 、左箭头或 `backspace`
前进	`v` 或右箭头
平移/缩放	`p`
缩放到矩形	`o`
保存	`ctrl + s`
切换全屏	`ctrl + f`
关闭绘图	`ctrl + w`
将平移/缩放限制于 `x` 轴	使用鼠标平移/缩放时按住 `x`
将平移/缩放限制于 `y` 轴	使用鼠标平移/缩放时按住 `y`
保留宽高比	使用鼠标平移/缩放时按住 `CONTROL`
切换网格	鼠标在轴域上时按下 `g`
切换 `x` 轴刻度（对数/线性）	鼠标在轴域上时按下 `L` 或 `k`
切换 `y` 轴刻度（对数/线性）	鼠标在轴域上时按下 `l`

如果你使用 `matplotlib.pyplot` ，则会为每个图形自动创建工具栏。 如果你正在编写自己的用户界面代码，则可以将工具栏添加为窗口小部件。 确切的语法取决于你的 UI，但在`matplotlib/examples/user_interfaces目录中有每个受支持的 UI 的示例。 这里是一些 GTK 的示例代码：

```
import gtk

from matplotlib.figure import Figure
from matplotlib.backends.backend_gtkagg import FigureCanvasGTKAg
g as FigureCanvas
from matplotlib.backends.backend_gtkagg import NavigationToolbar
2GTKAgg as NavigationToolbar

win = gtk.Window()
win.connect("destroy", lambda x: gtk.main_quit())
win.set_default_size(400,300)
win.set_title("Embedding in GTK")

vbox = gtk.VBox()
win.add(vbox)

fig = Figure(figsize=(5,4), dpi=100)
ax = fig.add_subplot(111)
ax.plot([1,2,3])

canvas = FigureCanvas(fig)  # a gtk.DrawingArea
vbox.pack_start(canvas)
toolbar = NavigationToolbar(canvas, win)
vbox.pack_start(toolbar, False, False)

win.show_all()
gtk.main()
```

在 Python shell 中使用 Matplotlib

> 原文：Using matplotlib in a python shell
>
> 译者：飞龙
>
>
>
> 警告
>
> 该页面的内容已严重过时。

默认情况下，matplotlib 将绘图延迟到脚本结束，因为绘图可能是开销大的操作，并且你可能不想在每次更改单个属性时更新绘图，而是只在所有属性更改后更新一次。

但是在 python shell 中工作时，通常需要用每个命令更新绘图，例如，在更改 `xlabel()` 或一行的标记样式之后。 虽然这在概念上很简单，但在实践中它可能很棘手，因为 matplotlib 在底层是一个图形用户界面应用程序，并拥有一些技巧，使应用程序在一个 python shell 正常工作。

使用 IPython 解决

> 注意
>
> 这里描述的模式出于历史原因仍然存在，但强烈建议不要使用。它污染函数的命名空间，会影响 python 内建设施，并可能导致错误难以跟踪。 要获得 IPython 集成而无需导入，使用 `%matplotlib` 魔术命令是首个选择。 参见 ipython 文档。

幸运的是，一个增强的交互式 python shell，ipython 已经找出了所有这些技巧，并且可被 matplotlib 感知，所以当你在 pylab 模式下启动 ipython。

```
johnh@flag:~> ipython
Python 2.4.5 (#4, Apr 12 2008, 09:09:16)
IPython 0.9.0 -- An enhanced Interactive Python.

In [1]: %pylab

  Welcome to pylab, a matplotlib-based Python environment.
  For more information, type 'help(pylab)'.

In [2]: x = randn(10000)

In [3]: hist(x, 100)
```

它为你设置一切使交互式绘图工作，就像你期望的那样。 调用 `figure()` 并弹出图形窗口，调用 `plot()` 使你的数据出现在图形窗口中。

注意在上面的例子中，我们没有导入任何 matplotlib 名称，因为在 pylab 模式下，ipython 将自动导入它们。 ipython 还为你启用交互模式，这会导致每个 pyplot 命令触发图形更新，并且还提供了一个 matplotlib 感知的运行命令，来高效运行 matplotlib 脚本。 ipython 在运行命令期间关闭交互模式，然后在运行结束时恢复交互状态，以便你可以手动继续调整图形。

ipython 已经嵌入了很多最近的作品，从 pylab 支持，到各种 GUI 应用程序，所以请检查 ipython 邮件列表的最新状态。

其它 Python 解释器

如果你不能使用 ipython，并且仍然想在交互式 python shell 使用 matplotlib/pylab，例如，plain-ole 标准的 python 交互式解释器，你将需要了解什么是 matplotlib 后端 。

有了 TkAgg 后端，它使用 Tkinter 用户界面工具包，你可以从任意的非 gui python shell 使用 matplotlib。 只需在你的 `matplotlibrc` 文件中设置 `backend : TkAgg` 和 `interactive : True` （请参阅自定义 matplotlib）并启动 python。 然后：

```
>>> from pylab import *
>>> plot([1,2,3])
>>> xlabel('hi mom')
```

应该能够开箱即用。 这也可能适用于最新版本的 qt4agg 和 gtkagg 后端，以及 Macintosh 上的 macosx 后端。 注意，在批处理模式下，即从脚本制作图形时，交互模式可能很慢，因为它用每个命令重绘图形。 因此，你可能需要仔细考虑，然后通过 `matplotlibrc` 文件而不是使用下一节中列出的函数，使其作为默认行为。

Gui shell 问题最多，因为它们必须运行主循环，但是交互式绘图也涉及主循环。 Ipython 已经为 matplotlib 主后端解决了这一切问题。 可能有其他 shell 和 IDE 也可以在交互模式下使用 matplotlib，但一个明显的候选项不会：python IDLE IDE 是一个不支持 pylab 交互模式的 Tkinter gui 应用程序，无论后端是什么。

控制交互式更新

`pyplot` 接口的 `interactive` 属性控制是否在每个 `pyplot` 命令上绘制图画布。 如果 `interactive` 是 `False` ，那么每个 `plot` 命令都会更新图形状态，但只会在显式调用 `draw()` 时绘制。 当 `interactive` 为 `True` 时，每个 `pyplot` 命令都会触发绘制。

`pyplot` 接口提供了 4 个有助于交互式控制的命令。

`isinteractive()`

返回交互式设置。 `True|False` 。

`ion()`

将交互式模式打开。

`ioff()`

将交互式模式关闭。

`draw()`

强制图形重新绘制。

当处理绘图开销很大的大型图形时，你可能希望临时关闭 matplotlib 的交互式设置来避免性能损失：

```
>>> #create big-expensive-figure
>>> ioff()      # turn updates off
>>> title('now how much would you pay?')
>>> xticklabels(fontsize=20, color='green')
>>> draw()      # force a draw
>>> savefig('alldone', dpi=300)
>>> close()
>>> ion()      # turn updating back on
>>> plot(rand(20), mfc='g', mec='r', ms=40, mew=4, ls='--', lw=3
)
```

事件处理及拾取

> 原文：Event handling and picking
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

matplotlib 使用了许多用户界面工具包（wxpython，tkinter，qt4，gtk 和 macosx），为了支持交互式平移和缩放图形等功能，拥有一套 API 通过按键和鼠标移动与图形交互，并且『GUI中立』，对开发人员十分有帮助，所以我们不必重复大量的代码来跨不同的用户界面。虽然事件处理 API 是 GUI 中立的，但它是基于 GTK 模型，这是 matplotlib 支持的第一个用户界面。与标准 GUI 事件相比，被触发的事件也比 matplotlib 丰富一些，例如包括发生事件的 `matplotlib.axes.Axes` 的信息。事件还能够理解 matplotlib 坐标系，并且在事件中以像素和数据坐标为单位报告事件位置。

事件连接

要接收事件，你需要编写一个回调函数，然后将你的函数连接到事件管理器，它是 `FigureCanvasBase` 的一部分。这是一个简单的例子，打印鼠标点击的位置和按下哪个按钮：

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(np.random.rand(10))

def onclick(event):
    print('button=%d, x=%d, y=%d, xdata=%f, ydata=%f' %
          (event.button, event.x, event.y, event.xdata, event.yd
ata))

cid = fig.canvas.mpl_connect('button_press_event', onclick)
```

`FigureCanvas` 的方法 `mpl_connect()` 返回一个连接 `id`，它只是一个整数。当你要断开回调时，只需调用：

```
fig.canvas.mpl_disconnect(cid)
```

> 注意
>
> 画布仅保留回调的弱引用。因此，如果回调是类实例的方法，你需要保留对该实例的引用。否则实例将被垃圾回收，回调将消失。

以下是可以连接到的事件，在事件发生时发回给你的类实例以及事件描述：

事件名称	类和描述
`'button_press_event'`	`MouseEvent` - 鼠标按钮被按下
`'button_release_event'`	`MouseEvent` - 鼠标按钮被释放
`'draw_event'`	`DrawEvent` - 画布绘图
`'key_press_event'`	`KeyEvent` - 按键被按下
`'key_release_event'`	`KeyEvent` - 按键被释放
`'motion_notify_event'`	`MouseEvent` - 鼠标移动
`'pick_event'`	`PickEvent` - 画布中的对象被选中
`'resize_event'`	`ResizeEvent` - 图形画布大小改变
`'scroll_event'`	`MouseEvent` - 鼠标滚轮被滚动
`'figure_enter_event'`	`LocationEvent` - 鼠标进入新的图形
`'figure_leave_event'`	`LocationEvent` - 鼠标离开图形
`'axes_enter_event'`	`LocationEvent` - 鼠标进入新的轴域
`'axes_leave_event'`	`LocationEvent` - 鼠标离开轴域

事件属性

所有 matplotlib 事件继承自基类 `matplotlib.backend_bases.Event`，储存以下属性：

`name`

事件名称

`canvas`

生成事件的 `FigureCanvas` 实例

`guiEvent`

触发 matplotlib 事件的 GUI 事件

最常见的事件是按键按下/释放事件、鼠标按下/释放和移动事件。 处理这些事件的 `KeyEvent` 和 `MouseEvent` 类都派生自 `LocationEvent`，它具有以下属性：

`x`

x 位置，距离画布左端的像素

`y`

y 位置，距离画布底端的像素

`inaxes`

如果鼠标经过轴域，则为 `Axes` 实例

`xdata`

鼠标的 `x` 坐标，以数据坐标为单位

`ydata`

鼠标的 `y` 坐标，以数据坐标为单位

但我们看一看画布的简单示例，其中每次按下鼠标时都会创建一条线段。

```
from matplotlib import pyplot as plt

class LineBuilder:
    def __init__(self, line):
        self.line = line
        self.xs = list(line.get_xdata())
        self.ys = list(line.get_ydata())
        self.cid = line.figure.canvas.mpl_connect('button_press_
event', self)

    def __call__(self, event):
        print('click', event)
        if event.inaxes!=self.line.axes: return
        self.xs.append(event.xdata)
        self.ys.append(event.ydata)
        self.line.set_data(self.xs, self.ys)
        self.line.figure.canvas.draw()

fig = plt.figure()
ax = fig.add_subplot(111)
ax.set_title('click to build line segments')
line, = ax.plot([0], [0])  # empty line
linebuilder = LineBuilder(line)

plt.show()
```

我们刚刚使用的 `MouseEvent` 是一个 `LocationEvent` ，因此我们可以访问 `event.x` 和 `event.xdata` 中的数据和像素坐标。除了 `LocationEvent` 属性，它拥有：

`button`

按下的按钮， `None` 、1、2、3、 `'up'` 、 `'down'` （ `'up'` 、 `'down'` 用于滚动事件）

`key`

按下的键， `None` ，任何字符， `'shift'` 、 `'win'` 或者 `'control'`

可拖拽的矩形练习

编写使用 `Rectangle` 实例初始化的可拖动矩形类，但在拖动时会移动其 `x` ， `y` 位置。 提示：你需要存储矩形的原始 `xy` 位置，存储为 `rect.xy` 并连接到按下，移动和释放鼠标事件。 当鼠标按下时，检查点击是否发生在你的矩形上（见 `matplotlib.patches.Rectangle.contains()` ），如果是，存储矩形 `xy` 和数据坐标为单位的鼠标点击位置。 在移动事件回调中，计算鼠标移动的 `deltax` 和 `deltay` ，并将这些增量添加到存储的原始矩形，并重新绘图。 在按钮释放事件中，只需将所有你存储的按钮按下数据重置为 `None` 。

这里是解决方案：

```python
import numpy as np
import matplotlib.pyplot as plt

class DraggableRectangle:
    def __init__(self, rect):
        self.rect = rect
        self.press = None

    def connect(self):
        'connect to all the events we need'
        self.cidpress = self.rect.figure.canvas.mpl_connect(
            'button_press_event', self.on_press)
        self.cidrelease = self.rect.figure.canvas.mpl_connect(
            'button_release_event', self.on_release)
        self.cidmotion = self.rect.figure.canvas.mpl_connect(
            'motion_notify_event', self.on_motion)

    def on_press(self, event):
        'on button press we will see if the mouse is over us and store some data'
        if event.inaxes != self.rect.axes: return

        contains, attrd = self.rect.contains(event)
        if not contains: return
        print('event contains', self.rect.xy)
        x0, y0 = self.rect.xy
        self.press = x0, y0, event.xdata, event.ydata

    def on_motion(self, event):
        'on motion we will move the rect if the mouse is over us'

        if self.press is None: return
        if event.inaxes != self.rect.axes: return
        x0, y0, xpress, ypress = self.press
        dx = event.xdata - xpress
```

```
        dy = event.ydata - ypress
        #print('x0=%f, xpress=%f, event.xdata=%f, dx=%f, x0+dx=%
f' %
        #      (x0, xpress, event.xdata, dx, x0+dx))
        self.rect.set_x(x0+dx)
        self.rect.set_y(y0+dy)

        self.rect.figure.canvas.draw()


    def on_release(self, event):
        'on release we reset the press data'
        self.press = None
        self.rect.figure.canvas.draw()

    def disconnect(self):
        'disconnect all the stored connection ids'
        self.rect.figure.canvas.mpl_disconnect(self.cidpress)
        self.rect.figure.canvas.mpl_disconnect(self.cidrelease)
        self.rect.figure.canvas.mpl_disconnect(self.cidmotion)

fig = plt.figure()
ax = fig.add_subplot(111)
rects = ax.bar(range(10), 20*np.random.rand(10))
drs = []
for rect in rects:
    dr = DraggableRectangle(rect)
    dr.connect()
    drs.append(dr)

plt.show()
```

附加题：使用动画秘籍中讨论的动画 blit 技术，使动画绘制更快更流畅。

附加题解决方案：

```
# draggable rectangle with the animation blit techniques; see
# http://www.scipy.org/Cookbook/Matplotlib/Animations
import numpy as np
import matplotlib.pyplot as plt

class DraggableRectangle:
    lock = None  # only one can be animated at a time
    def __init__(self, rect):
        self.rect = rect
        self.press = None
        self.background = None

    def connect(self):
        'connect to all the events we need'
        self.cidpress = self.rect.figure.canvas.mpl_connect(
```

```
            'button_press_event', self.on_press)
        self.cidrelease = self.rect.figure.canvas.mpl_connect(
            'button_release_event', self.on_release)
        self.cidmotion = self.rect.figure.canvas.mpl_connect(
            'motion_notify_event', self.on_motion)

    def on_press(self, event):
        'on button press we will see if the mouse is over us and
 store some data'
        if event.inaxes != self.rect.axes: return
        if DraggableRectangle.lock is not None: return
        contains, attrd = self.rect.contains(event)
        if not contains: return
        print('event contains', self.rect.xy)
        x0, y0 = self.rect.xy
        self.press = x0, y0, event.xdata, event.ydata
        DraggableRectangle.lock = self

        # draw everything but the selected rectangle and store t
he pixel buffer
        canvas = self.rect.figure.canvas
        axes = self.rect.axes
        self.rect.set_animated(True)
        canvas.draw()
        self.background = canvas.copy_from_bbox(self.rect.axes.b
box)

        # now redraw just the rectangle
        axes.draw_artist(self.rect)

        # and blit just the redrawn area
        canvas.blit(axes.bbox)

    def on_motion(self, event):
        'on motion we will move the rect if the mouse is over us'

        if DraggableRectangle.lock is not self:
            return
        if event.inaxes != self.rect.axes: return
        x0, y0, xpress, ypress = self.press
        dx = event.xdata - xpress
        dy = event.ydata - ypress
        self.rect.set_x(x0+dx)
        self.rect.set_y(y0+dy)

        canvas = self.rect.figure.canvas
        axes = self.rect.axes
        # restore the background region
        canvas.restore_region(self.background)

        # redraw just the current rectangle
        axes.draw_artist(self.rect)
```

```
        # blit just the redrawn area
        canvas.blit(axes.bbox)

    def on_release(self, event):
        'on release we reset the press data'
        if DraggableRectangle.lock is not self:
            return

        self.press = None
        DraggableRectangle.lock = None

        # turn off the rect animation property and reset the bac
kground
        self.rect.set_animated(False)
        self.background = None

        # redraw the full figure
        self.rect.figure.canvas.draw()

    def disconnect(self):
        'disconnect all the stored connection ids'
        self.rect.figure.canvas.mpl_disconnect(self.cidpress)
        self.rect.figure.canvas.mpl_disconnect(self.cidrelease)
        self.rect.figure.canvas.mpl_disconnect(self.cidmotion)

fig = plt.figure()
ax = fig.add_subplot(111)
rects = ax.bar(range(10), 20*np.random.rand(10))
drs = []
for rect in rects:
    dr = DraggableRectangle(rect)
    dr.connect()
    drs.append(dr)

plt.show()
```

鼠标进入和离开

如果希望在鼠标进入或离开图形时通知你，你可以连接到图形/轴域进入/离开事件。 下面是一个简单的例子，它改变了鼠标所在的轴域和图形的背景颜色：

```
"""
Illustrate the figure and axes enter and leave events by changin
g the
frame colors on enter and leave
"""
import matplotlib.pyplot as plt

def enter_axes(event):
    print('enter_axes', event.inaxes)
    event.inaxes.patch.set_facecolor('yellow')
    event.canvas.draw()

def leave_axes(event):
    print('leave_axes', event.inaxes)
    event.inaxes.patch.set_facecolor('white')
    event.canvas.draw()

def enter_figure(event):
    print('enter_figure', event.canvas.figure)
    event.canvas.figure.patch.set_facecolor('red')
    event.canvas.draw()

def leave_figure(event):
    print('leave_figure', event.canvas.figure)
    event.canvas.figure.patch.set_facecolor('grey')
    event.canvas.draw()

fig1 = plt.figure()
fig1.suptitle('mouse hover over figure or axes to trigger events'
)
ax1 = fig1.add_subplot(211)
ax2 = fig1.add_subplot(212)

fig1.canvas.mpl_connect('figure_enter_event', enter_figure)
fig1.canvas.mpl_connect('figure_leave_event', leave_figure)
fig1.canvas.mpl_connect('axes_enter_event', enter_axes)
fig1.canvas.mpl_connect('axes_leave_event', leave_axes)

fig2 = plt.figure()
fig2.suptitle('mouse hover over figure or axes to trigger events'
)
ax1 = fig2.add_subplot(211)
ax2 = fig2.add_subplot(212)

fig2.canvas.mpl_connect('figure_enter_event', enter_figure)
fig2.canvas.mpl_connect('figure_leave_event', leave_figure)
fig2.canvas.mpl_connect('axes_enter_event', enter_axes)
fig2.canvas.mpl_connect('axes_leave_event', leave_axes)

plt.show()
```

对象拾取

你可以通过设置艺术家的 `picker` 属性（例如，matplotlib `Line2D` ， `Text` ， `Patch` ， `Polygon` ， `AxesImage` 等）来启用选择，

`picker` 属性有多种含义：

`None`

选择对于该艺术家已禁用（默认）

`boolean`

如果为 `True` ，则启用选择，当鼠标移动到该艺术家上方时，会触发事件

`float`

如果选择器是数字，则将其解释为点的 ε 公差，并且如果其数据在鼠标事件的 ε 内，则艺术家将触发事件。 对于像线条和补丁集合的一些艺术家，艺术家可以向生成的选择事件提供附加数据，例如，在选择事件的 ε 内的数据的索引。

函数

如果拾取器是可调用的，则它是用户提供的函数，用于确定艺术家是否被鼠标事件击中。 签名为 `hit, props = picker(artist, mouseevent)` ，用于测试是否命中。 如果鼠标事件在艺术家上，返回 `hit = True` ， `props` 是一个属性字典，它们会添加到 `PickEvent` 属性。

通过设置 `picker` 属性启用对艺术家进行拾取后，你需要连接到图画布的 `pick_event` ，以便在鼠标按下事件中获取拾取回调。 例如：

```
def pick_handler(event):
    mouseevent = event.mouseevent
    artist = event.artist
    # now do something with this...
```

传给你的回调的 `PickEvent` 事件永远有两个属性：

`mouseevent`

是生成拾取事件的鼠标事件。鼠标事件具有像 `x` 和 `y` （显示空间中的坐标，例如，距离左，下的像素）和 `xdata` ， `ydata` （数据空间中的坐标）的属性。 此外，你可以获取有关按下哪些按钮，按下哪些键，鼠标在哪个轴域上面等信息。详细信息请参阅 `matplotlib.backend_bases.MouseEvent` 。

`artist`

生成拾取事件的 `Artist` 。

另外，像 `Line2D` 和 `PatchCollection` 的某些艺术家可以将附加的元数据（如索引）附加到满足选择器标准的数据中（例如，行中在指定 ε 容差内的所有点）

简单拾取示例

在下面的示例中，我们将行选择器属性设置为标量，因此它表示以点为单位的容差（72 点/英寸）。当拾取事件位于距离线条的容差范围时，将调用 `onpick` 回调函数，并且带有在拾取距离容差内的数据顶点索引。我们的 `onpick` 回调函数只打印在拾取位置上的数据。不同的 matplotlib 艺术家可以将不同的数据附加到 `PickEvent`。例如，`Line2D` 将 `ind` 属性作为索引附加到拾取点下面的行数据中。有关 `Line` 的 `PickEvent` 属性的详细信息，请参阅 `pick()`。这里是代码：

```
import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111)
ax.set_title('click on points')

line, = ax.plot(np.random.rand(100), 'o', picker=5)  # 5 points
tolerance

def onpick(event):
    thisline = event.artist
    xdata = thisline.get_xdata()
    ydata = thisline.get_ydata()
    ind = event.ind
    points = tuple(zip(xdata[ind], ydata[ind]))
    print('onpick points:', points)

fig.canvas.mpl_connect('pick_event', onpick)

plt.show()
```

拾取练习

创建含有 100 个数组的数据集，包含 1000 个高斯随机数，并计算每个数组的样本平均值和标准差（提示：`numpy` 数组具有 `mean` 和 `std` 方法），并制作 100 个均值与 100 个标准的 `xy` 标记图。将绘图命令创建的线条连接到拾取事件，并绘制数据的原始时间序列，这些数据生成了被点击的点。如果在被点击的点的容差范围内存在多于一个点，则可以使用多个子图来绘制多个时间序列。

练习的解决方案：

```
"""
compute the mean and stddev of 100 data sets and plot mean vs st
ddev.
When you click on one of the mu, sigma points, plot the raw data
 from
the dataset that generated the mean and stddev
"""
import numpy as np
import matplotlib.pyplot as plt

X = np.random.rand(100, 1000)
xs = np.mean(X, axis=1)
ys = np.std(X, axis=1)

fig = plt.figure()
ax = fig.add_subplot(111)
ax.set_title('click on point to plot time series')
line, = ax.plot(xs, ys, 'o', picker=5)  # 5 points tolerance


def onpick(event):

    if event.artist!=line: return True

    N = len(event.ind)
    if not N: return True


    figi = plt.figure()
    for subplotnum, dataind in enumerate(event.ind):
        ax = figi.add_subplot(N,1,subplotnum+1)
        ax.plot(X[dataind])
        ax.text(0.05, 0.9, 'mu=%1.3f\nsigma=%1.3f'%(xs[dataind],
 ys[dataind]),
                transform=ax.transAxes, va='top')
        ax.set_ylim(-0.5, 1.5)
    figi.show()
    return True

fig.canvas.mpl_connect('pick_event', onpick)

plt.show()
```

所选示例

屏幕截图

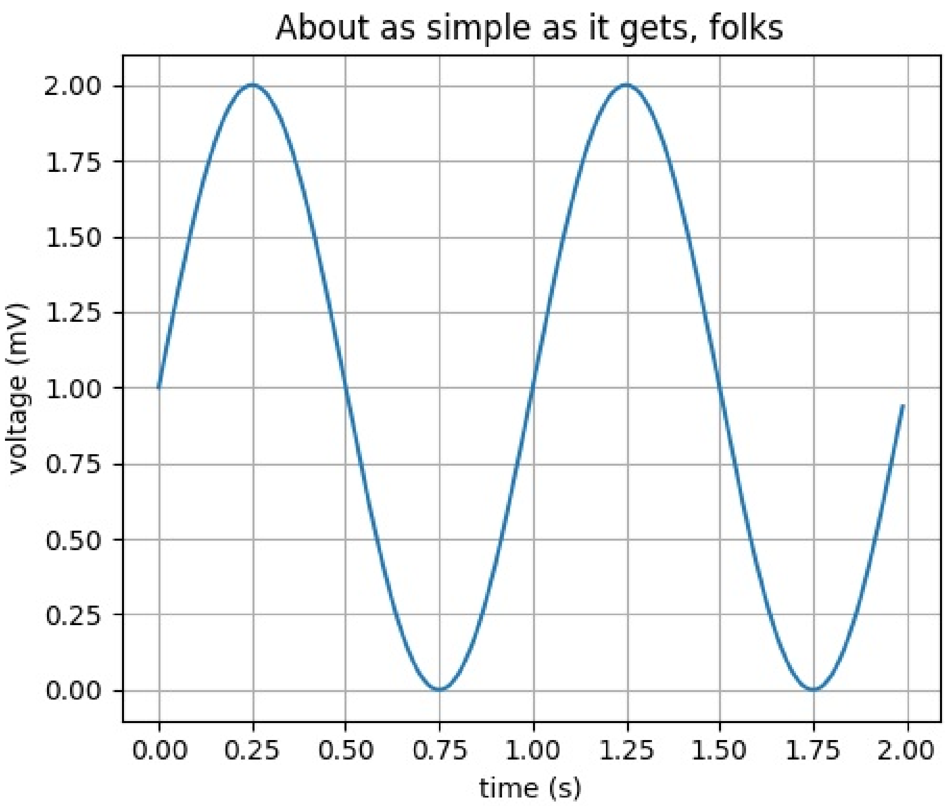
> 原文：Screenshots
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

这里你会找到一些示例图和生成它们的代码。

简单绘图

这里是一个带有文本标签的基本的绘图：

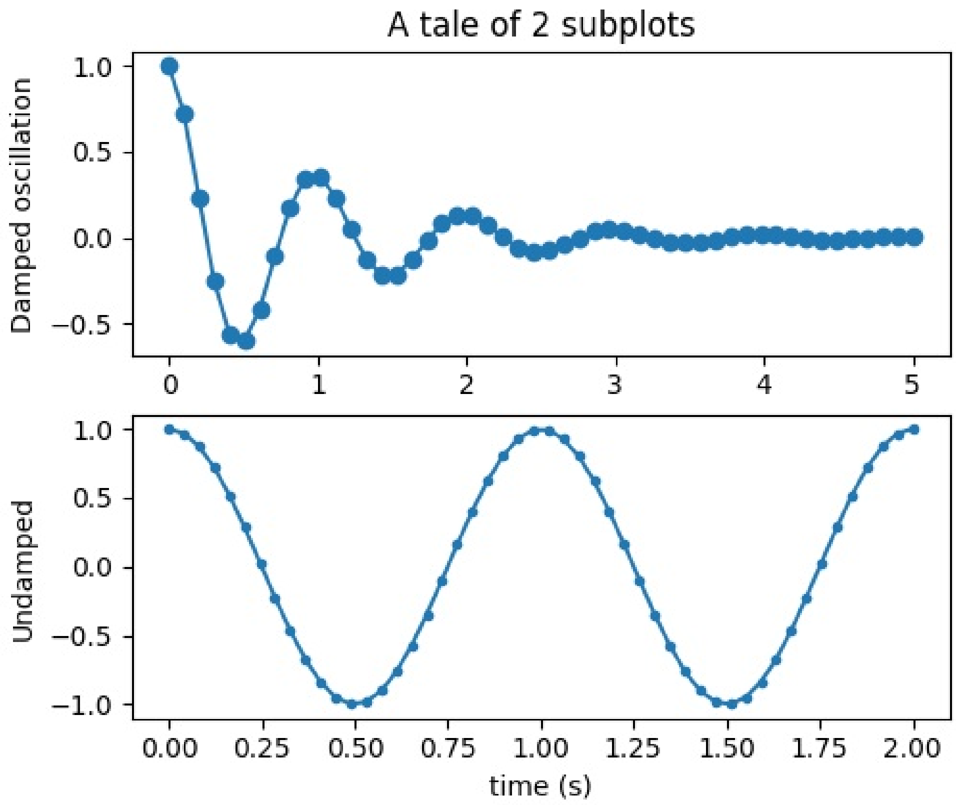
源代码



子图示例

多个轴域（例如子图）可使用 `subplot()` 命令创建：

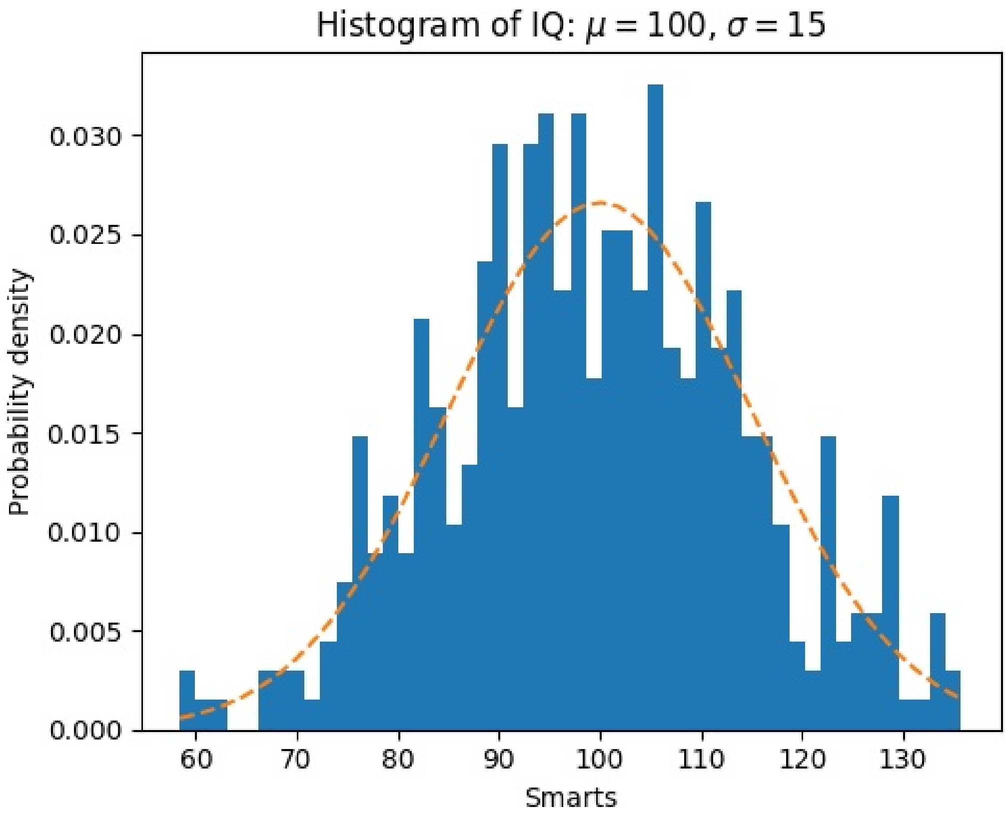
源代码



直方图

`hist()` 命令自动生成直方图，并返回项数或者概率：

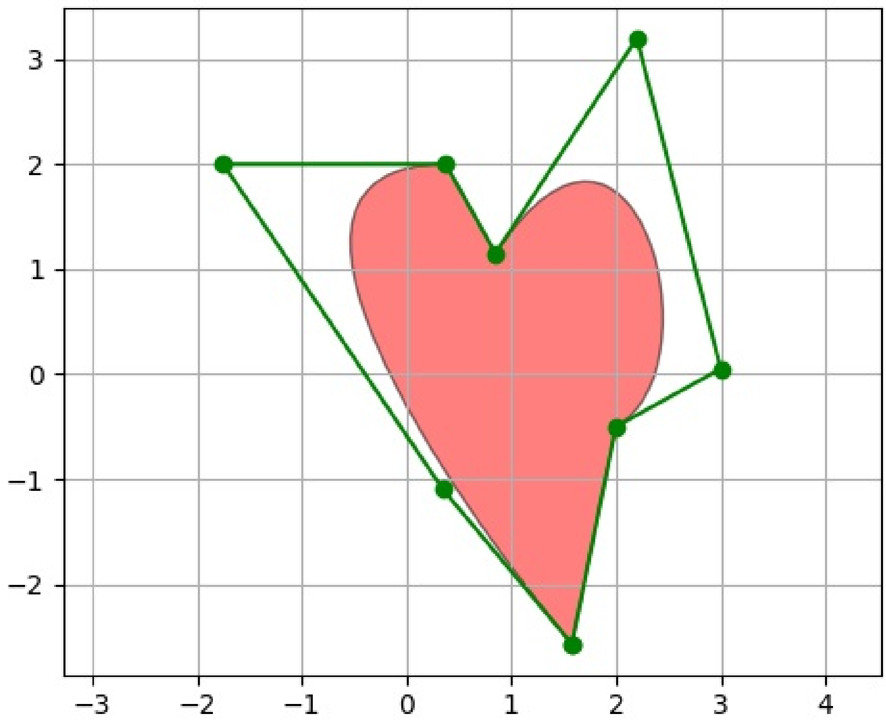
源代码



路径示例

你可以使用 `matplotlib.path` 模块，在 `maplotlib` 中添加任意路径：

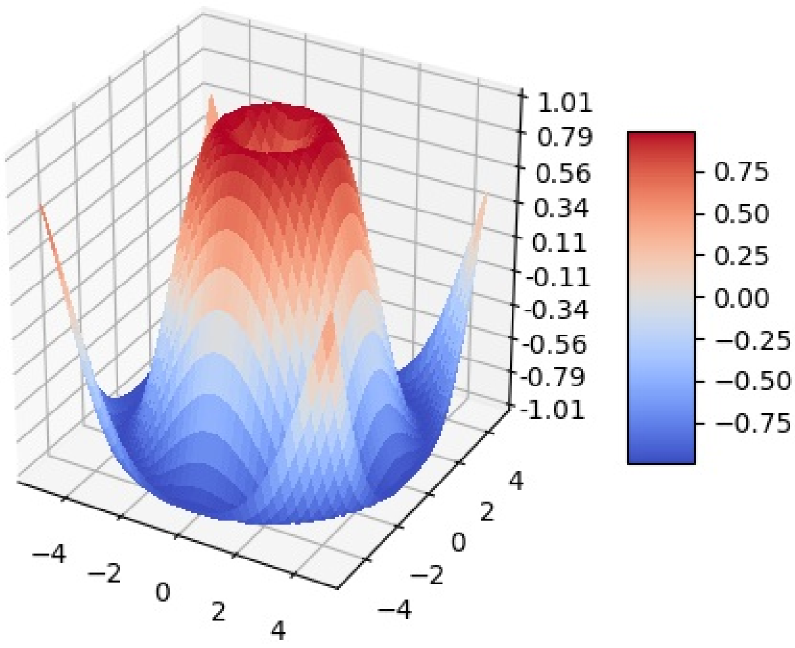
源代码



mplot3d

mplot3d 工具包（见 mplot3d 教程和 mplot3d 示例）支持简单的三维图形，包括平台、线框图、散点图和条形图。

源代码

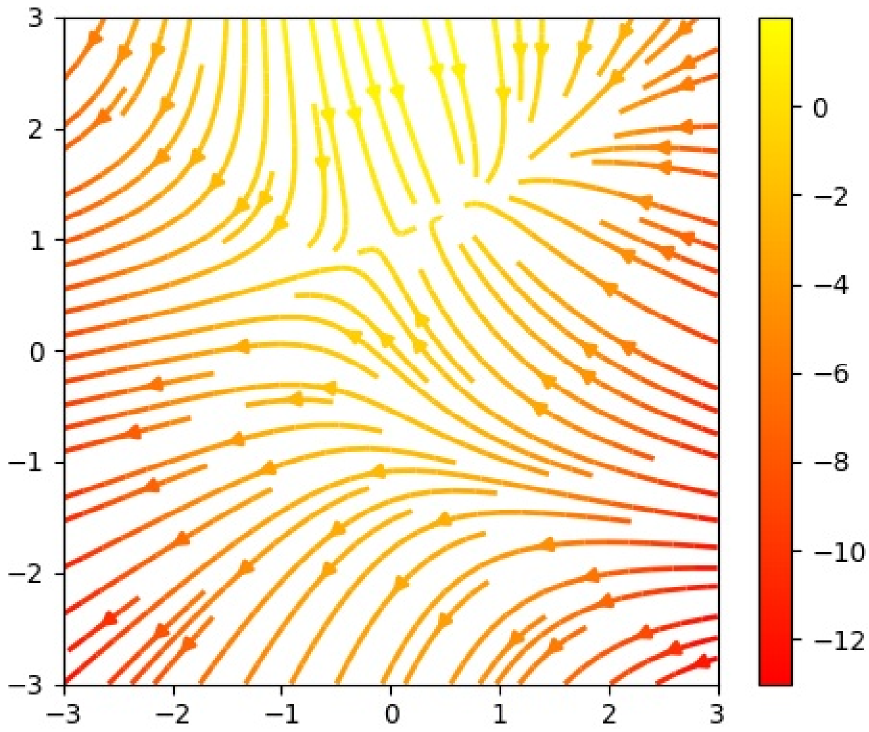


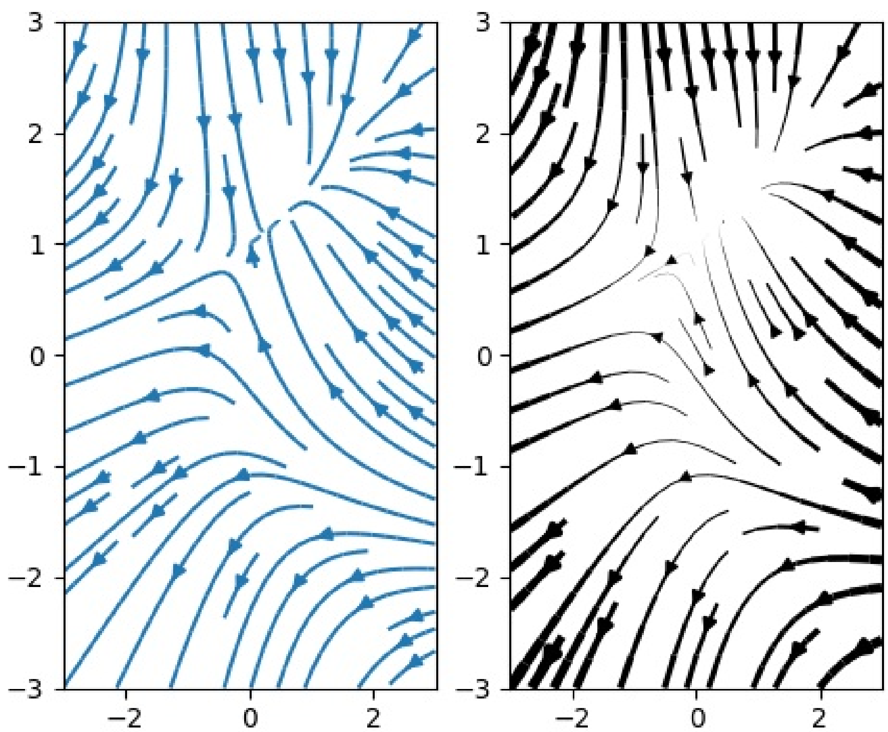
感谢 John Porter，Jonathan Taylor，Reinier Heeres 和 Ben Root 开发了 mplot3d 工具包。此工具包包含于所有标准 matplotlib 安装中。

Streamplot

`streamplot()` 函数绘制向量场的流线图。除了简单地绘制流线之外，它还允许将流线的颜色和/或线宽映射到单独的参数，例如向量场的速度或局部密度。

源代码



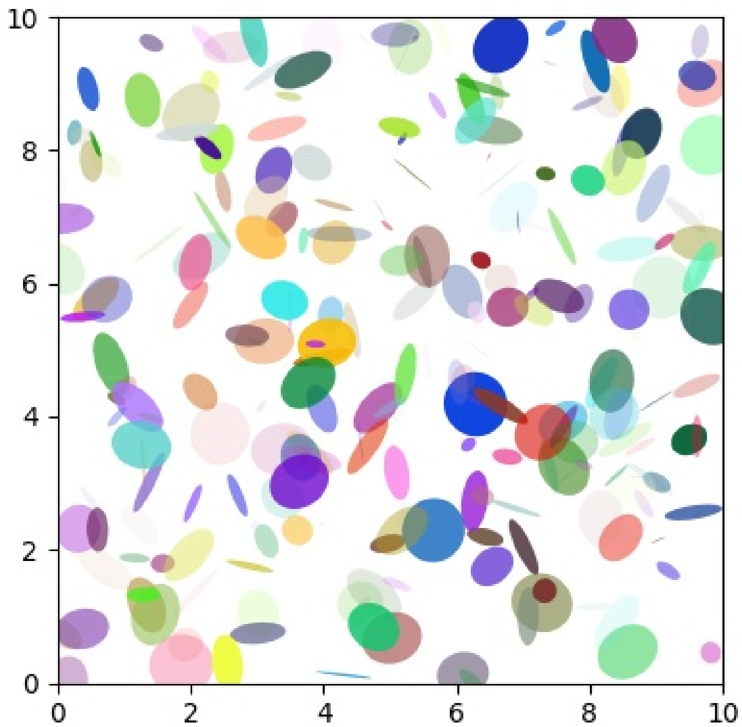


这个特性完善了绘制向量场的 `quiver()` 函数。感谢 Tom Flanagan 和 Tony You 添加 `streamplot` 函数。

椭圆

为了支持 Phoenix Mars Mission（使用 matplotlib 展示地面跟踪的航天器），Michael Droettboom 在 Charlie Moad 的工作基础上提供了非常精确的椭圆弧的 8-样条近似（见 `Arc`），它对缩放级别并不敏感。

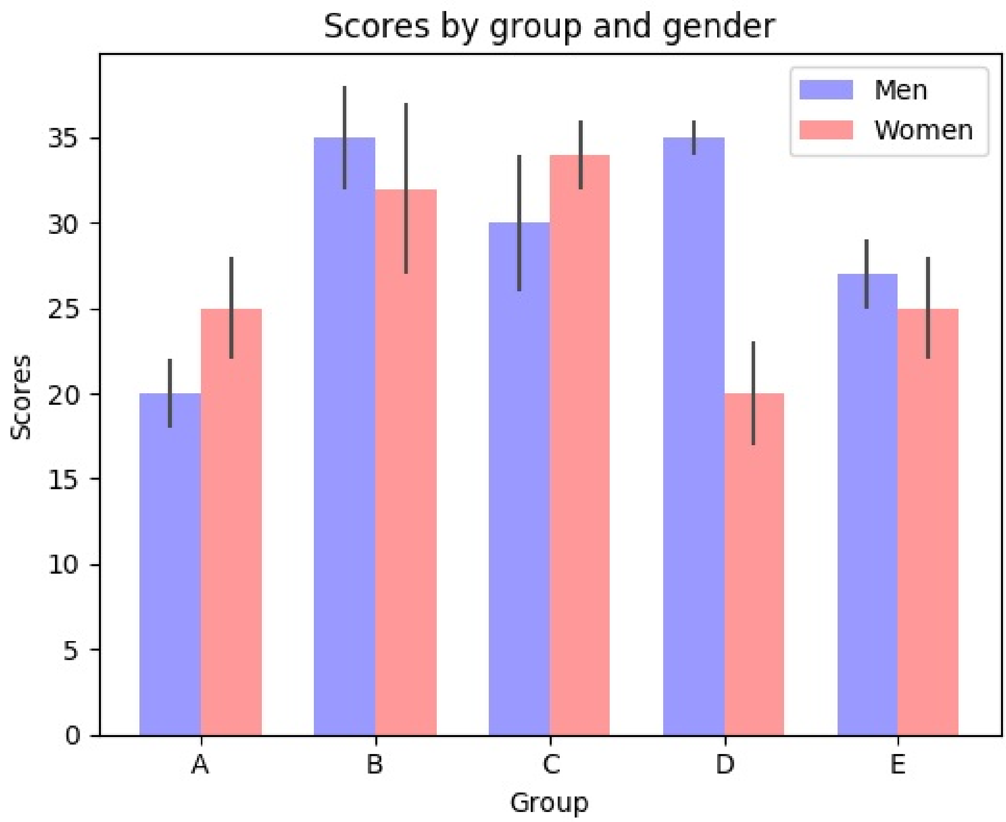
源代码



条形图

使用 `bar()` 命令创建条形图十分容易，其中包括一些定制（如误差条）：

源代码

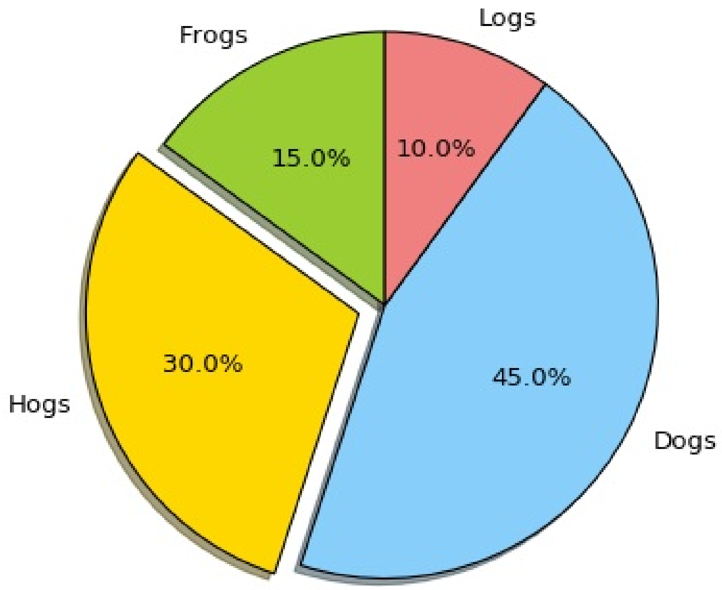


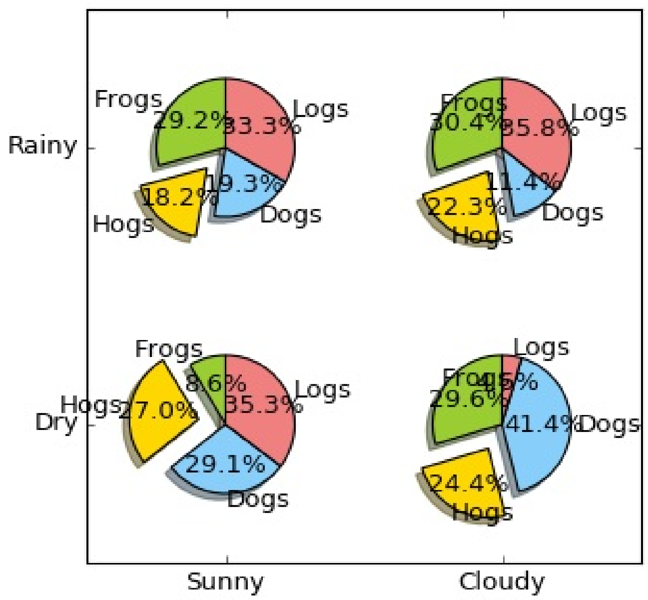
创建堆叠条（ `bar_stacked.py` ），蜡烛条（ `finance_demo.py` ）和水平条形图（ `barh_demo.py` ）也很简单。

饼图

`pie()` 命令允许您轻松创建饼图。 可选功能包括自动标记区域的百分比，从饼图中心向外生成一个或多个楔形以及阴影效果。 仔细查看附加的代码，它用几行代码来生成这个图像。

源代码

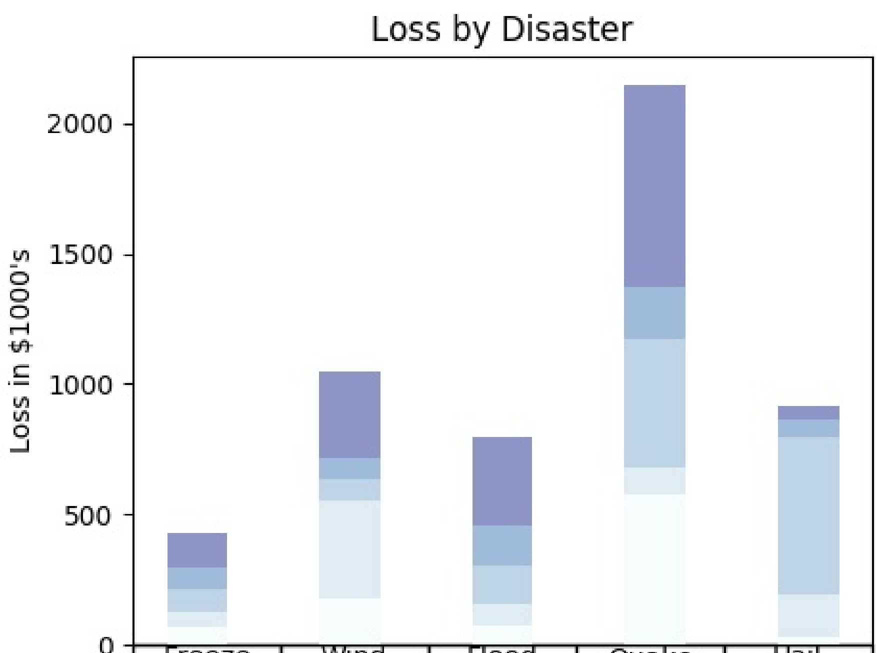




表格示例

`table()` 命令向轴域添加文本表格。

源代码

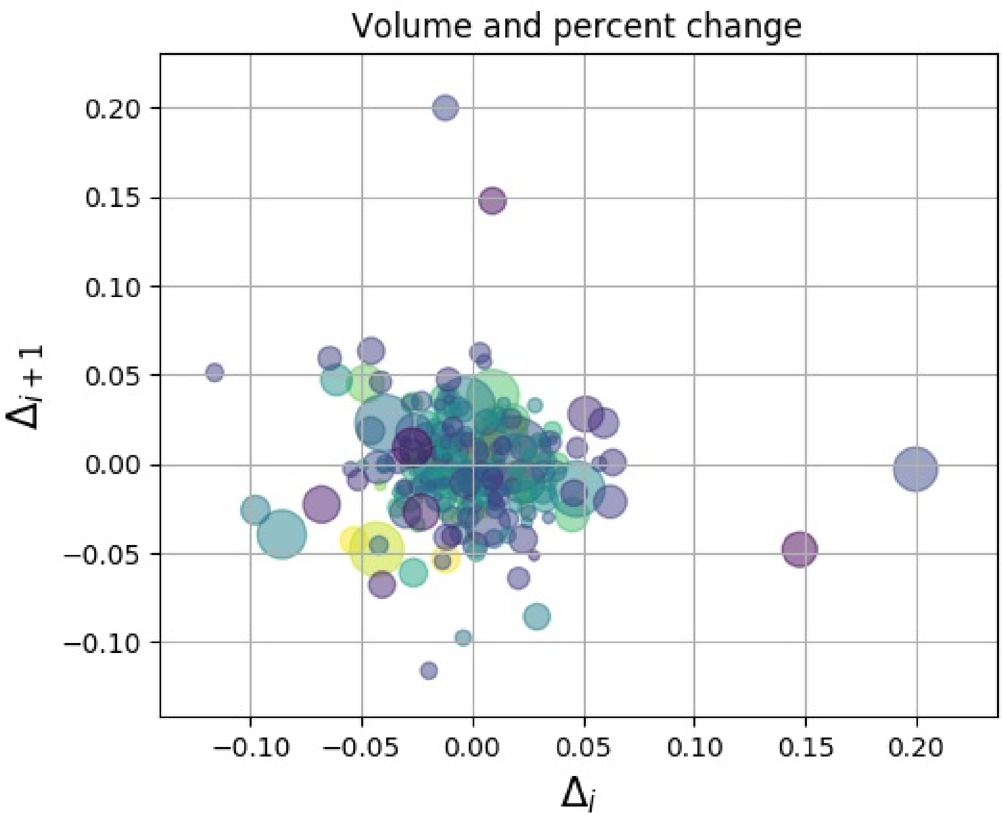


	Freeze	Wind	Flood	Quake	Hail
100 year	431.5	1049.4	799.6	2149.8	917.9
50 year	292.2	717.8	456.4	1368.5	865.6
20 year	213.8	636.0	305.7	1175.2	796.0
10 year	124.6	555.4	153.2	677.2	192.5
5 year	66.4	174.3	75.1	577.9	32.0

散点图示例

`scatter()` 命令使用（可选的）大小和颜色参数创建散点图。 此示例描绘了 Google 股票价格的变化，标记的尺寸反映了交易量，并且颜色随时间变化。 这里，ALPHA 属性用于制作半透明圆形标记。

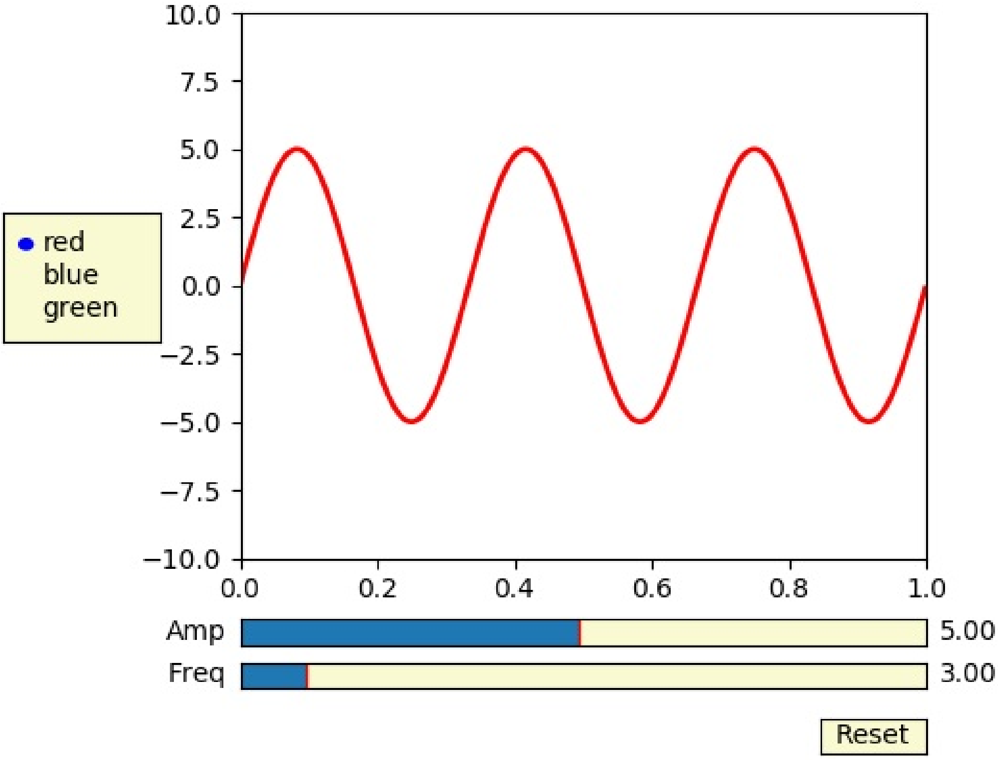
源代码



滑块示例

Matplotlib 拥有基本的 GUI 小部件，它们独立于您正在使用的图形用户界面，允许您编写 GUI 交叉图形和小部件。请参阅 `matplotlib.widgets` 和小部件示例。

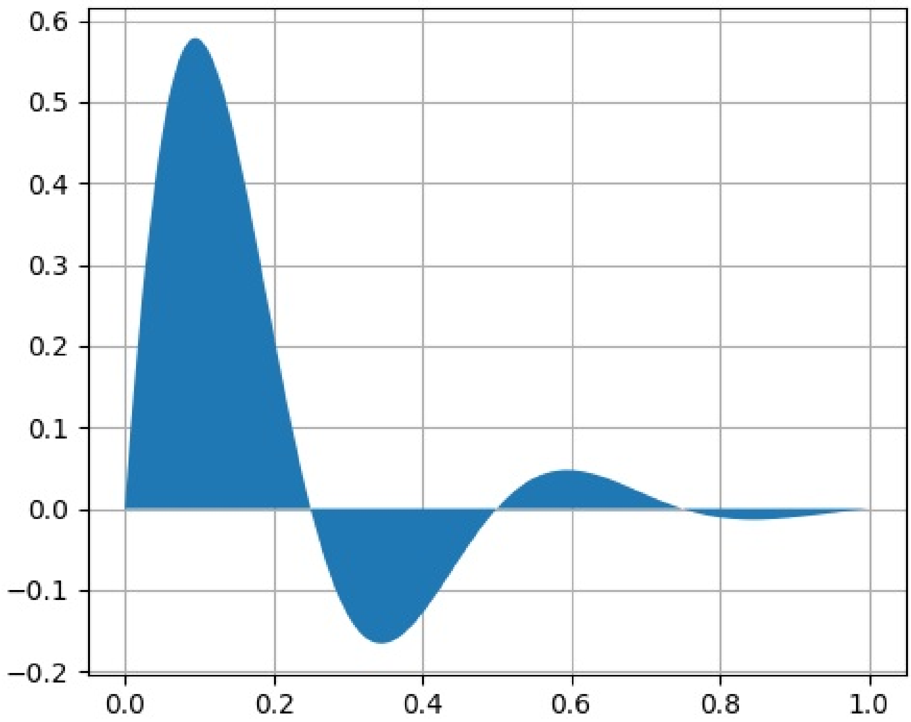
源代码



填充示例

`fill()` 命令可以绘制填充曲线和多边形：

源代码



感谢 Andrew Straw 添加了这个函数。

日期示例

您可以绘制日期数据与主要和次要刻度，以及用于二者的自定义刻度格式化器。

源代码

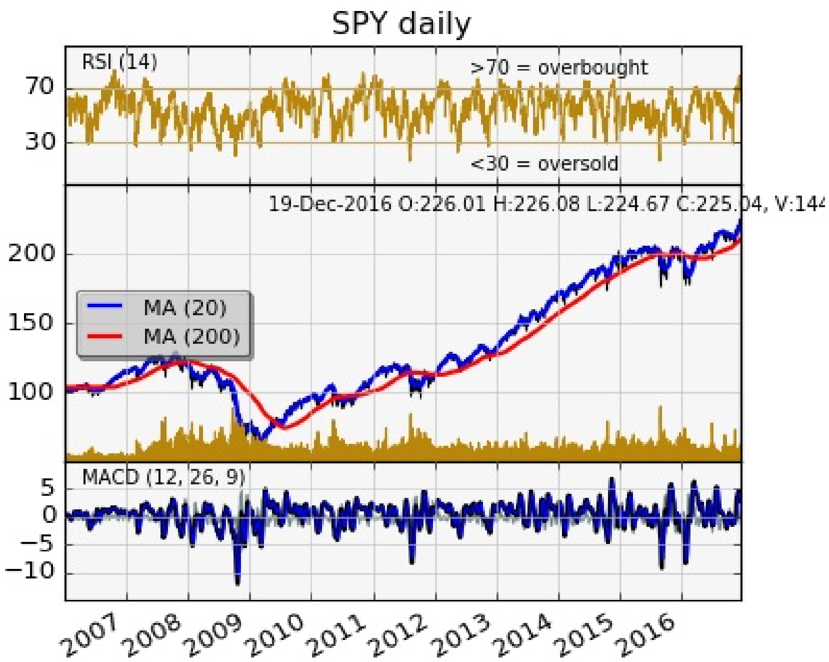


详细信息和用法请参阅 `matplotlib.ticker` 和 `matplotlib.dates` 。

金融图表

您可以通过结合 matplotlib 提供的各种绘图函数，布局命令和标签工具来创建复杂的金融图表。 以下示例模拟 ChartDirector 中的一个财务图：

源代码



地图示例

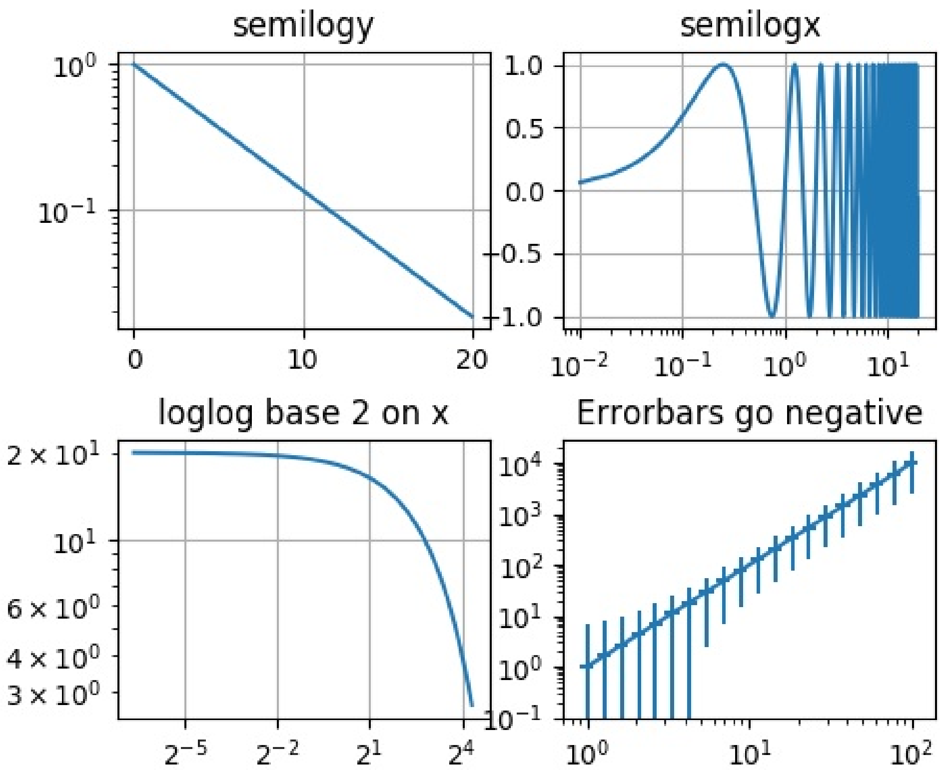
Jeff Whitaker 的 Basemap 附加工具包可以在许多不同的地图投影上绘制数据。 此示例展示了如何在直角投影上绘制轮廓，标记和文本，以 NASA 的“蓝色大理石”卫星图像作为背景。

源代码

对数绘图

`semilogx()` ， `semilogy()` 和 `loglog()` 函数简化了对数绘图的创建。

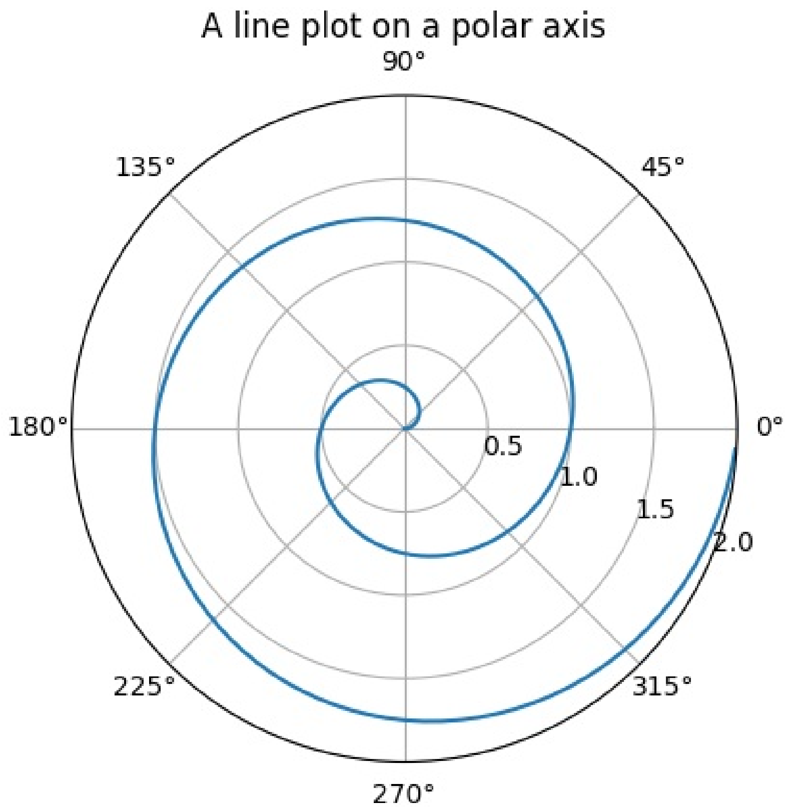
源代码



极轴绘图

`polar()` 命令生成极轴绘图。

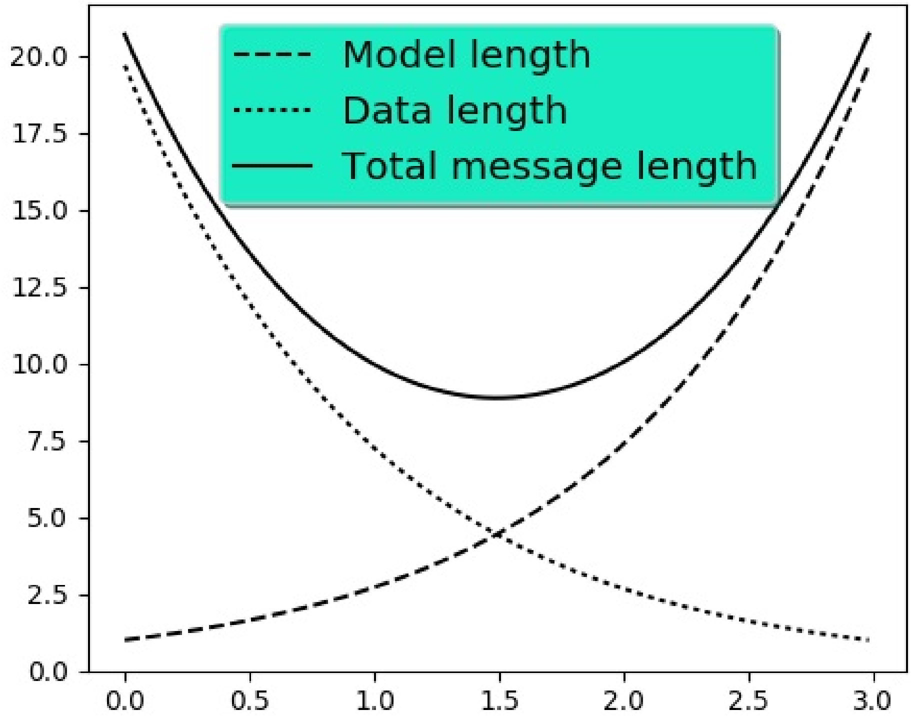
源代码



图例

`legend()` 命令使用 MATLAB 兼容的图例布局命令自动生成图形图例。

源代码

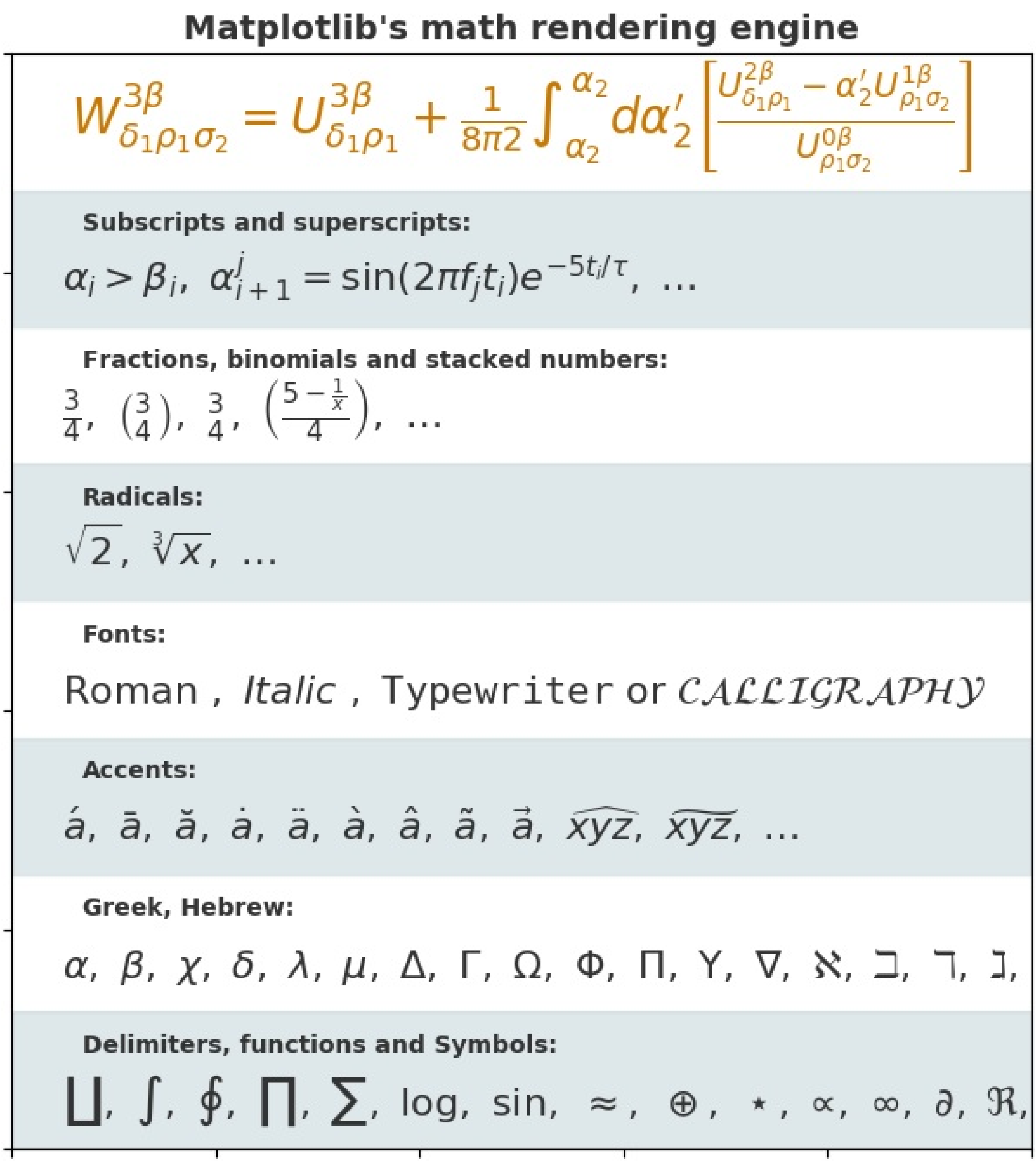


感谢 Charles Twardy 编写了图例命令的输入。

数学公式示例

下面是 matplotlib 内部数学公式引擎现在支持的许多 TeX 表达式的示例。`mathtext` 模块使用 freetype2 和 BaKoMa 或 STIX 现代字体提供 TeX 风格的数学表达式。其他详细信息请参阅 `matplotlib.mathtext` 模块。

源代码

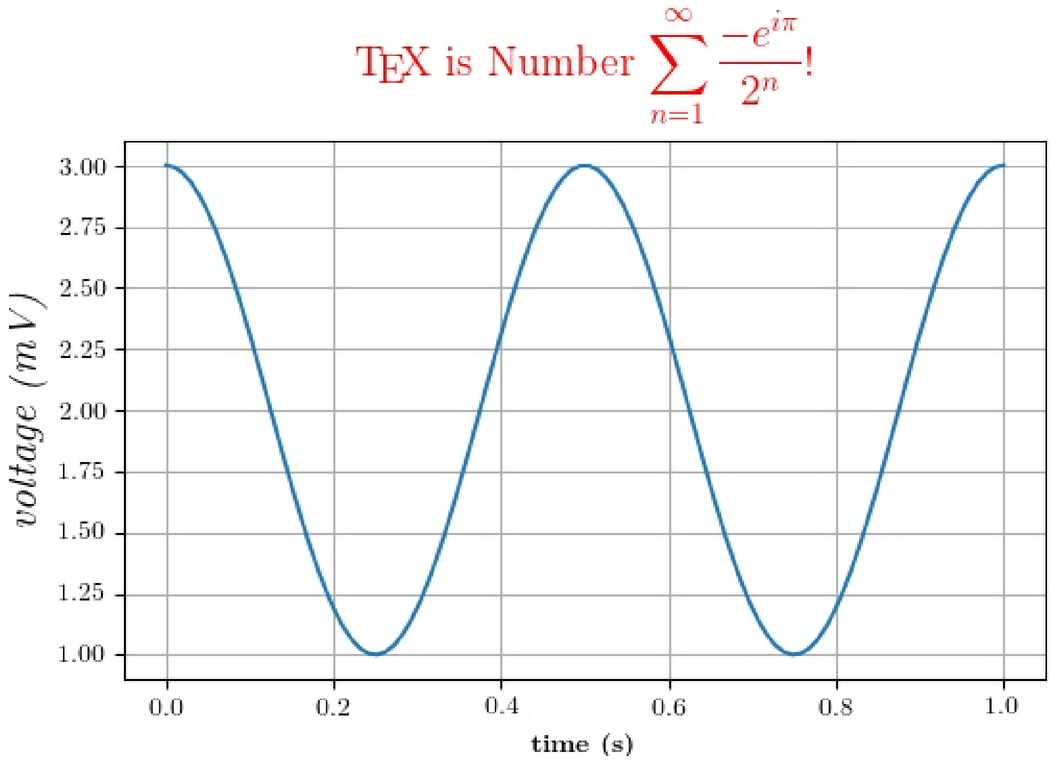


Matplotlib 的 `mathtext` 基础结构是一个独立的实现，不需要 TeX 或计算机上安装的任何外部软件包。请参阅编写数学表达式教程。

TeX 原生渲染

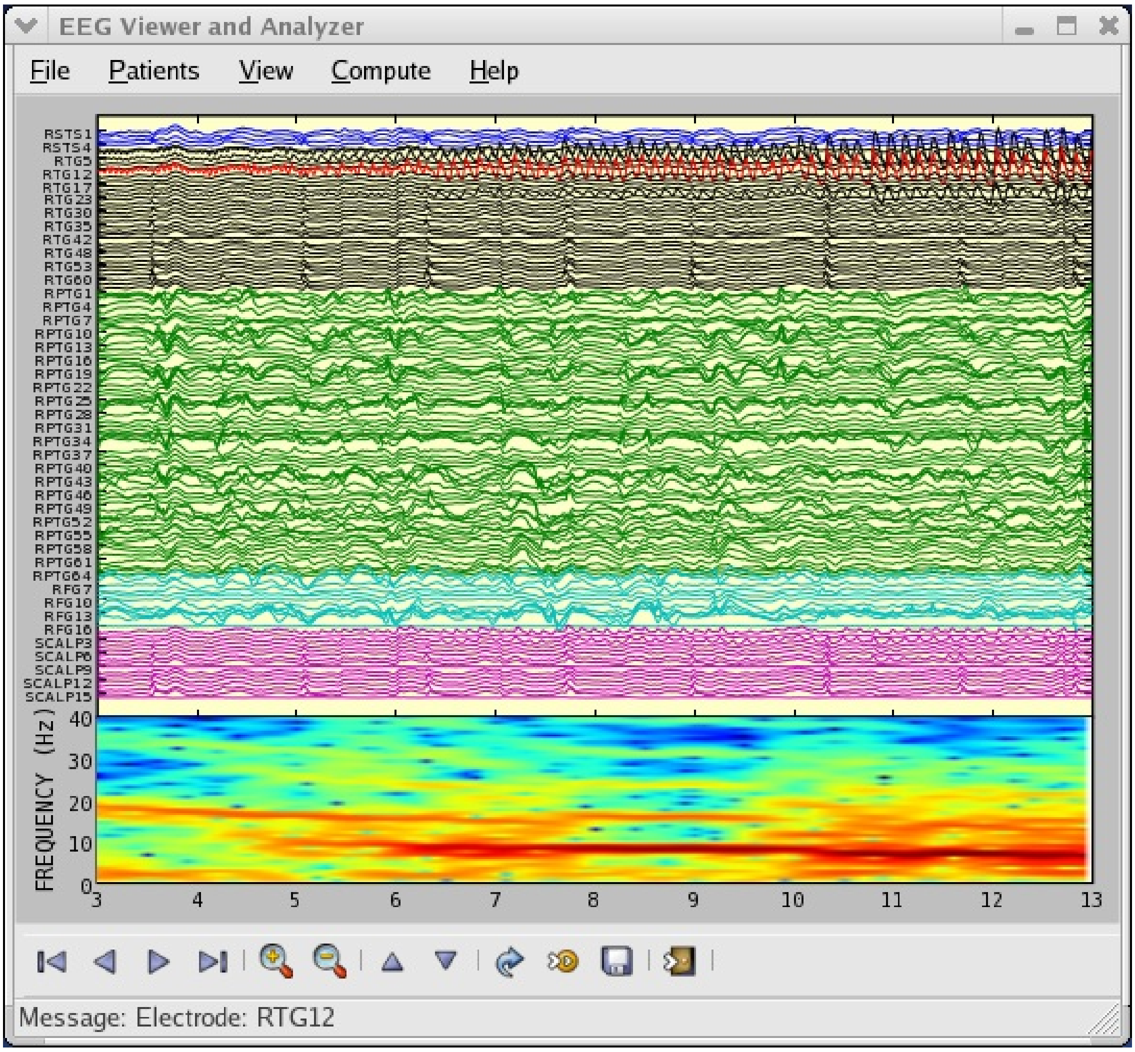
虽然 matplotlib 的内部数学渲染引擎相当强大，但有时你还是需要 TeX。Matplotlib 支持带有 usetex 选项的 TeX 外部字符串渲染。

源代码



EEG 示例

您可以将 matplotlib 嵌入到 pygtk，wx，Tk，FLTK 或 Qt 应用程序中。 这是一个名为 pbrain 的 EEG 查看器的屏幕截图。



下轴使用 `specgram()` 绘制其中一个 EEG 通道的频谱图。

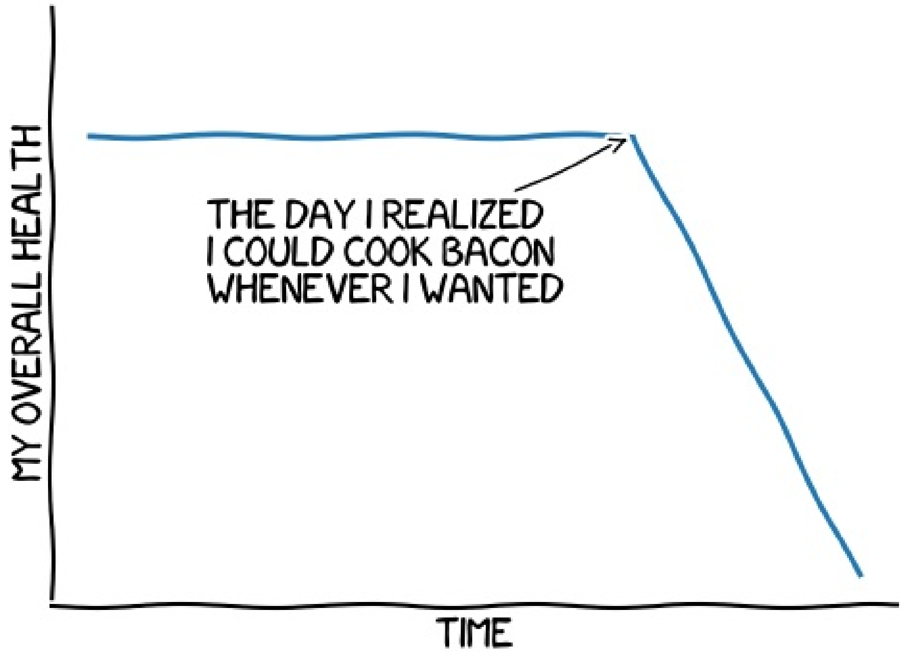
有关将 matplotlib 嵌入不同工具包的示例，请参阅：

- `user_interfaces` 示例代码：`embedding_in_gtk2.py`
- `user_interfaces` 示例代码：`embedding_in_wx2.py`
- `user_interfaces` 示例代码：`mpl_with_glade.py`
- `user_interfaces` 示例代码：`embedding_in_qt4.py`
- `user_interfaces` 示例代码：`embedding_in_tk.py`

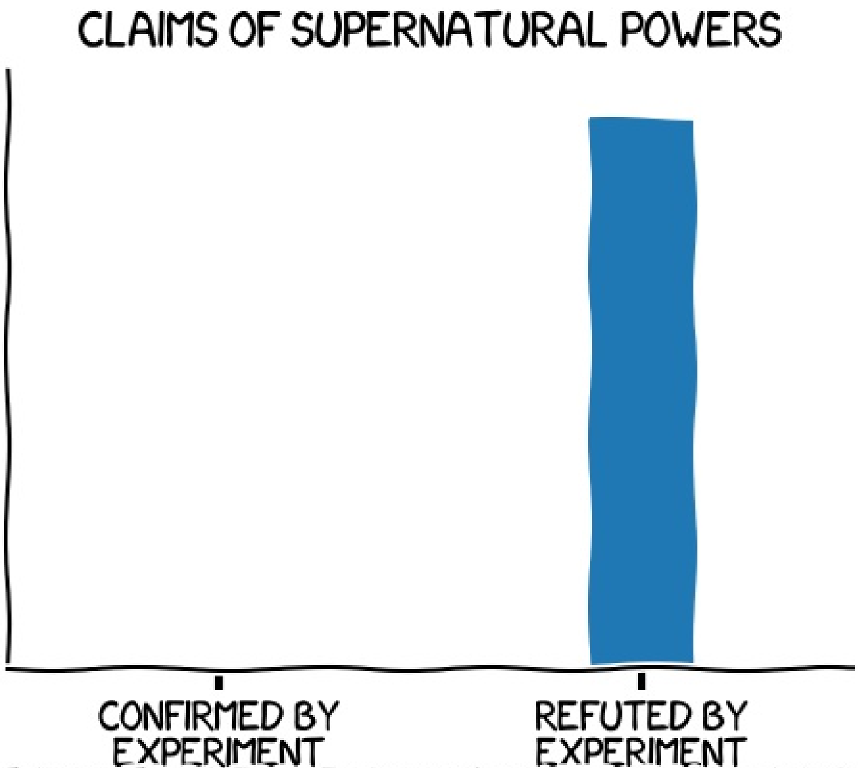
XKCD 风格的手绘图

matplotlib 支持 `xkcd` 风格的绘图。

源代码



"STOVE OWNERSHIP" FROM XKCD BY RANDALL MONROE



"THE DATA SO FAR" FROM XKCD BY RANDALL MONROE

我们最喜欢的秘籍

> 原文：Our Favorite Recipes
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

这里是一个简短的教程，示例和代码片段的集合，展示了一些有用的经验和技巧，来制作更精美的图像，并克服一些 matplotlib 的缺陷。

共享轴限制和视图

通常用于使两个或更多绘图共享一个轴，例如，两个子绘图具有时间作为公共轴。当你平移和缩放一个绘图，你想让另一个绘图一起移动。为了方便这一点，matplotlib 轴支持 `sharex` 和 `sharey` 属性。创建 `subplot()` 或 `axes()` 实例时，你可以传入一个关键字，表明要共享的轴。

```
In [96]: t = np.arange(0, 10, 0.01)

In [97]: ax1 = plt.subplot(211)

In [98]: ax1.plot(t, np.sin(2*np.pi*t))
Out[98]: [<matplotlib.lines.Line2D object at 0x98719ec>]

In [99]: ax2 = plt.subplot(212, sharex=ax1)

In [100]: ax2.plot(t, np.sin(4*np.pi*t))
Out[100]: [<matplotlib.lines.Line2D object at 0xb7d8fec>]
```

轻松创建子图

在 matplotlib 的早期版本中，如果你想使用 pythonic API 并创建一个 `figure` 实例，并从中创建一个 `subplots` 网格，而且可能带有共享轴，它涉及大量的样板代码。例如：

```
# old style
fig = plt.figure()
ax1 = fig.add_subplot(221)
ax2 = fig.add_subplot(222, sharex=ax1, sharey=ax1)
ax3 = fig.add_subplot(223, sharex=ax1, sharey=ax1)
ax3 = fig.add_subplot(224, sharex=ax1, sharey=ax1)
```

Fernando Perez 提供了一个很好的顶级方法，来一次性创建 `subplots()` （注意末尾的 `s` ），并为所有子图开启 `x` 和 `y` 共享。 你可以单独解构来获取轴域：

```
# new style method 1; unpack the axes
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, sharex=True,
sharey=True)
ax1.plot(x)
```

或将它们作为行数乘列数的对象数组返回，支持 numpy 索引：

```
# new style method 2; use an axes array
fig, axs = plt.subplots(2, 2, sharex=True, sharey=True)
axs[0,0].plot(x)
```

修复常见的日期问题

matplotlib 允许你本地绘制 python datetime 实例，并且在大多数情况下，可以很好地挑选刻度位置和字符串格式。 但有几件事情它不能妥善处理，这里有一些技巧，用于帮助你解决他们。 我们将在 `numpy` 记录数组中加载一些包含 `datetime.date` 对象的示例日期数据：

```
In [63]: datafile = cbook.get_sample_data('goog.npy')

In [64]: r = np.load(datafile).view(np.recarray)

In [65]: r.dtype
Out[65]: dtype([('date', '|O4'), ('', '|V4'), ('open', '<f8'),
                ('high', '<f8'), ('low', '<f8'), ('close', '<f8'
),
                ('volume', '<i8'),  ('adj_close', '<f8')])

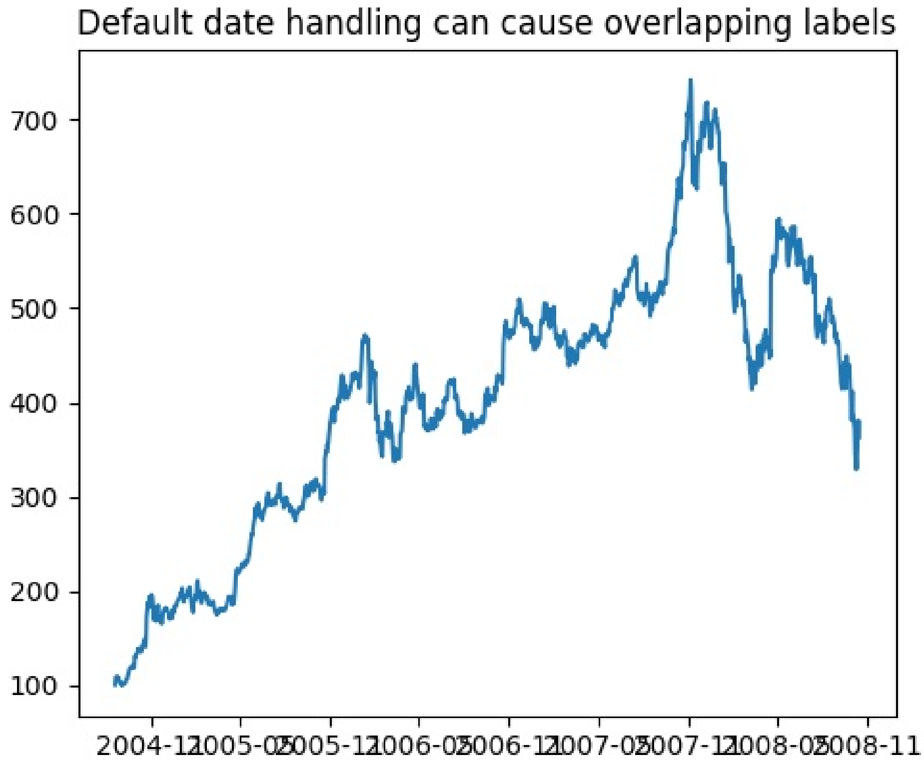
In [66]: r.date
Out[66]:
array([2004-08-19, 2004-08-20, 2004-08-23, ..., 2008-10-10, 2008
-10-13,
       2008-10-14], dtype=object)
```

字段日期的 `numpy` 记录数组的 `dtype` 是 `| O4` ，这意味着它是一个 4 字节的 python 对象指针; 在这种情况下，对象是 `datetime.date` 实例，当我们在 ipython 终端窗口中打印一些样本时，我们可以看到。

如果你绘制数据，

```
In [67]: plot(r.date, r.close)
Out[67]: [<matplotlib.lines.Line2D object at 0x92a6b6c>]
```

你会看到 x 轴标签重合到一起。

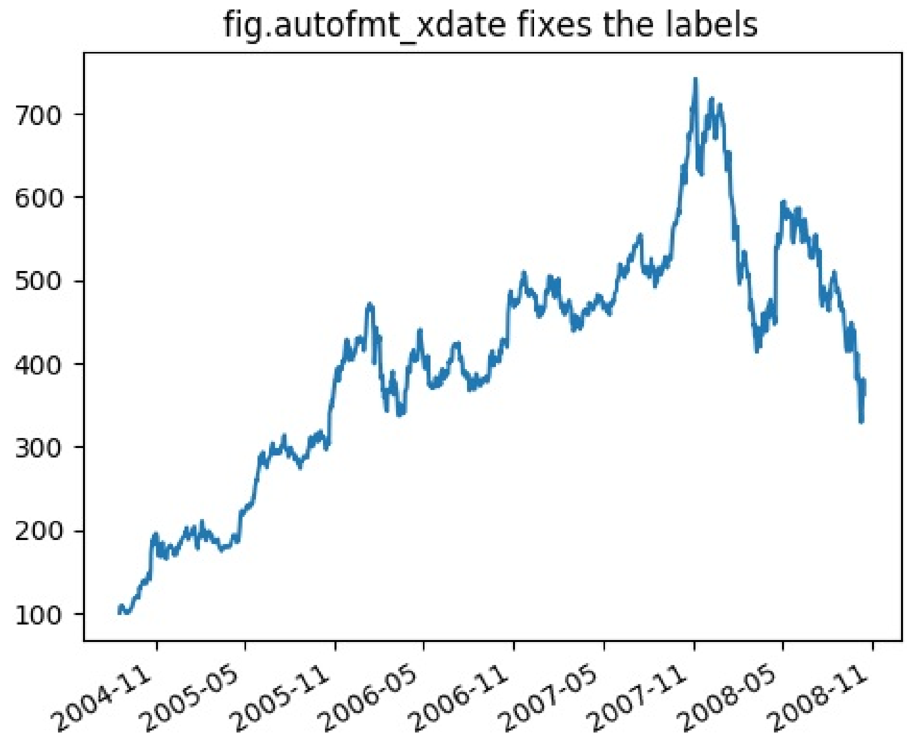


另一个麻烦是，如果你将鼠标悬停在窗口上，并在 x 和 y 坐标处查看 matplotlib 工具栏（交互式导航）的右下角，你会看到 x 位置的格式与刻度标签的格式相同， 例如，『Dec 2004』。 我们想要的是工具栏中的位置具有更高的精确度，例如，鼠标悬停在上面时给我们确切的日期。 为了解决第一个问题，我们可以使用 `matplotlib.figure.Figure.autofmt_xdate()` 。修复第二个问题，我们可以使用 `ax.fmt_xdata` 属性，该属性可以设置为任何接受标量并返回字符串的函数。 matplotlib 有一些内置的日期格式化器，所以我们将使用其中的一个。

```
plt.close('all')
fig, ax = plt.subplots(1)
ax.plot(r.date, r.close)

# rotate and align the tick labels so they look better
fig.autofmt_xdate()

# use a more precise date string for the x axis locations in the
# toolbar
import matplotlib.dates as mdates
ax.fmt_xdata = mdates.DateFormatter('%Y-%m-%d')
plt.title('fig.autofmt_xdate fixes the labels')
```

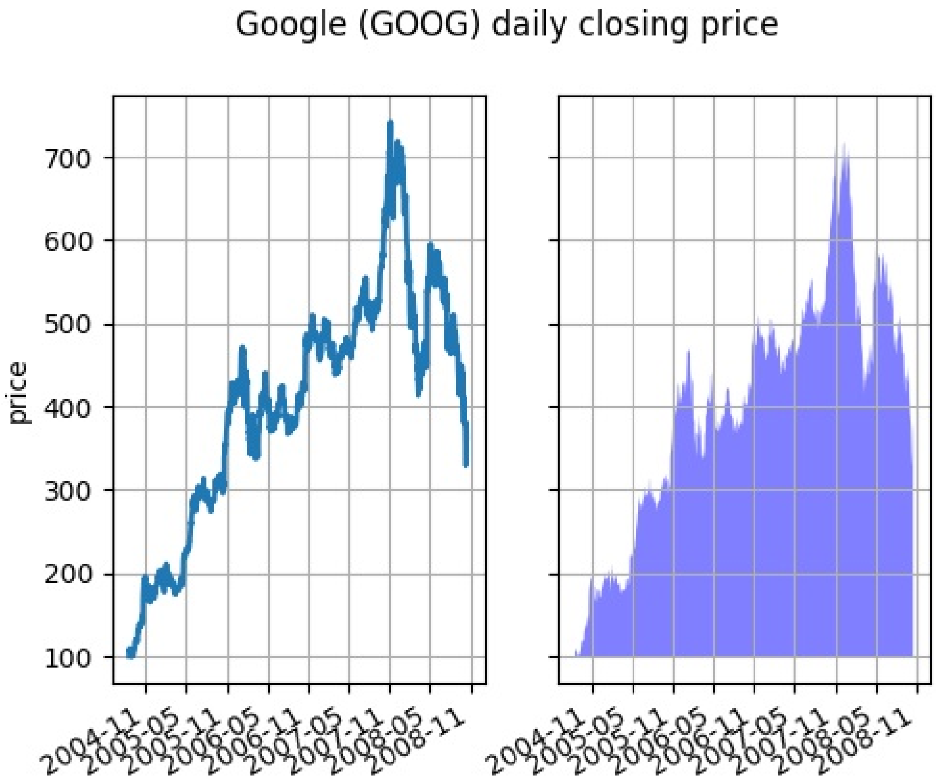


现在，当你将鼠标悬停在绘制的数据上，你将在工具栏中看到如`2004-12-01`的日期格式字符串。

透明度填充

`fill_between()`函数在最小和最大边界之间生成阴影区域，用于展示范围。 它有一个非常方便的参数，将填充范围与逻辑范围组合，例如，以便仅填充超过某个阈值的曲线。=

基本上，`fill_between`可以用来增强图形的视觉外观。 让我们比较两个财务-时间图表，左边是一个简单的线框图，右边是一个填充图。



Alpha 通道在这里不是必需的，但它可以用来软化颜色，创建更具视觉吸引力的绘图。 在其他示例中，我们将在下面看到，Alpha 通道在功能上有用，因为阴影区域可以重叠，Alpha 允许你同时看到两者。 注意，postscript 格式不支持 alpha（这是一个 postscript 限制，而不是一个 matplotlib 限制），因此，当使用 alpha 时，将你的数字保存在 PNG，PDF 或 SVG 中。

我们的下一个例子是计算随机漫步的两个群体，它们具有不同的正态分布平均值和标准差，足迹会从中绘制。我们使用共享区域来绘制群体的平均位置的加/减一个标准差。 这里的 Alpha 通道是有用的，不只是为了审美。

```
import matplotlib.pyplot as plt
import numpy as np

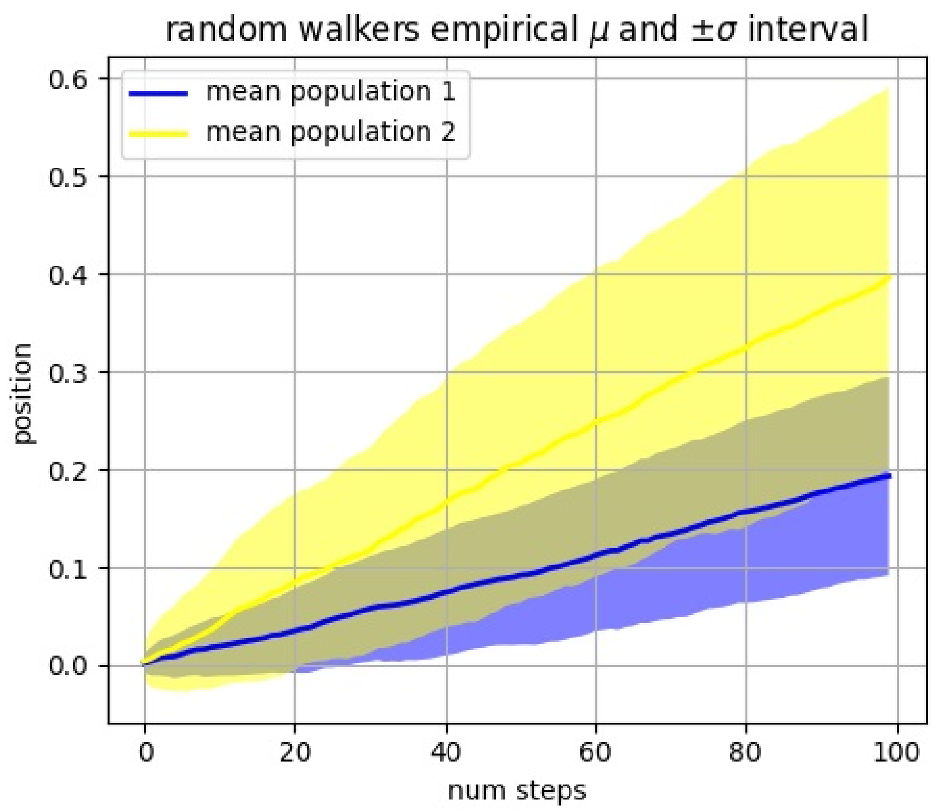
Nsteps, Nwalkers = 100, 250
t = np.arange(Nsteps)

# an (Nsteps x Nwalkers) array of random walk steps
S1 = 0.002 + 0.01*np.random.randn(Nsteps, Nwalkers)
S2 = 0.004 + 0.02*np.random.randn(Nsteps, Nwalkers)

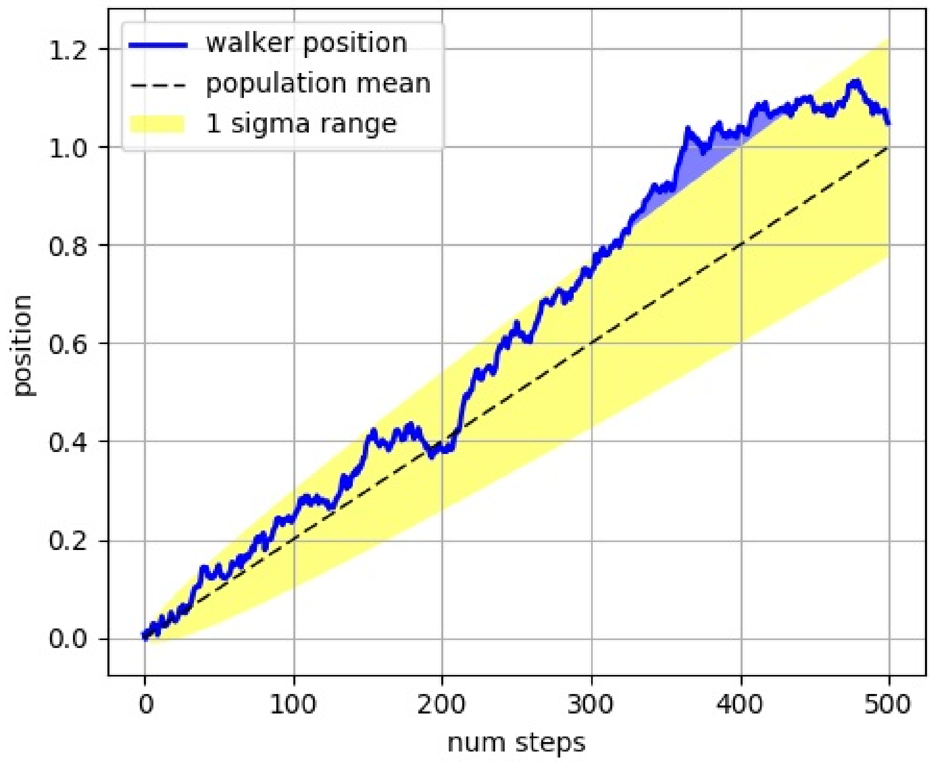
# an (Nsteps x Nwalkers) array of random walker positions
X1 = S1.cumsum(axis=0)
X2 = S2.cumsum(axis=0)


# Nsteps length arrays empirical means and standard deviations o
f both
# populations over time
mu1 = X1.mean(axis=1)
sigma1 = X1.std(axis=1)
mu2 = X2.mean(axis=1)
sigma2 = X2.std(axis=1)

# plot it!
fig, ax = plt.subplots(1)
ax.plot(t, mu1, lw=2, label='mean population 1', color='blue')
ax.plot(t, mu2, lw=2, label='mean population 2', color='yellow')
ax.fill_between(t, mu1+sigma1, mu1-sigma1, facecolor='blue', alp
ha=0.5)
ax.fill_between(t, mu2+sigma2, mu2-sigma2, facecolor='yellow', a
lpha=0.5)
ax.set_title('random walkers empirical $\mu$ and $\pm \sigma$ in
terval')
ax.legend(loc='upper left')
ax.set_xlabel('num steps')
ax.set_ylabel('position')
ax.grid()
```



`where` 关键字参数非常方便地用于突出显示图形的某些区域。其中使用与 `x` ，`ymin` 和 `ymax` 参数相同长度的布尔掩码，并且只填充布尔掩码为 `True` 的区域。在下面的例子中，我们模拟一个随机漫步者，并计算人口位置的分析平均值和标准差。群体平均值显示为黑色虚线，并且平均值的加/减一个标准差显示为黄色填充区域。我们使用 `where=X>upper_bound` 找到漫步者在一个标准差边界之上的区域，并将该区域变成蓝色。



透明、花式图例

有时你在绘制数据之前就知道你的数据是什么样的，并且可能知道例如右上角没有太多数据。 然后，你可以安全地创建不覆盖你的数据的图例：

```
ax.legend(loc='upper right')
```

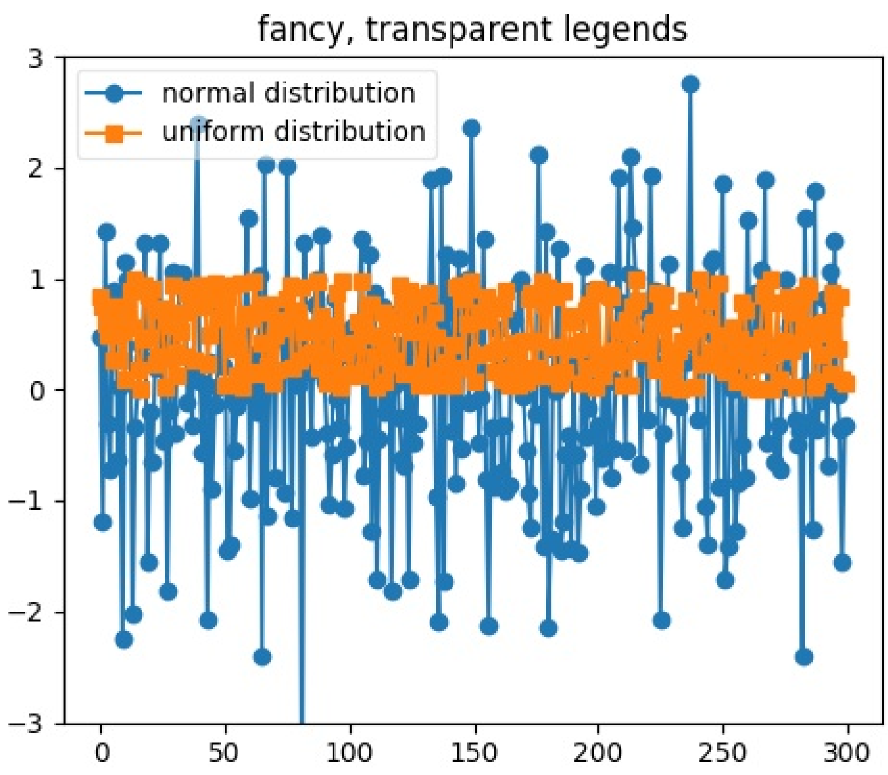
其他时候你不知道你的数据在哪里，而 `loc ='best'` 将尝试和放置图例：

```
ax.legend(loc='best')
```

但仍然，你的图例可能会覆盖你的数据，在这些情况下，使图例框架透明非常不错。

```
np.random.seed(1234)
fig, ax = plt.subplots(1)
ax.plot(np.random.randn(300), 'o-', label='normal distribution')
ax.plot(np.random.rand(300), 's-', label='uniform distribution')
ax.set_ylim(-3, 3)
ax.legend(loc='best', fancybox=True, framealpha=0.5)

ax.set_title('fancy, transparent legends')
```



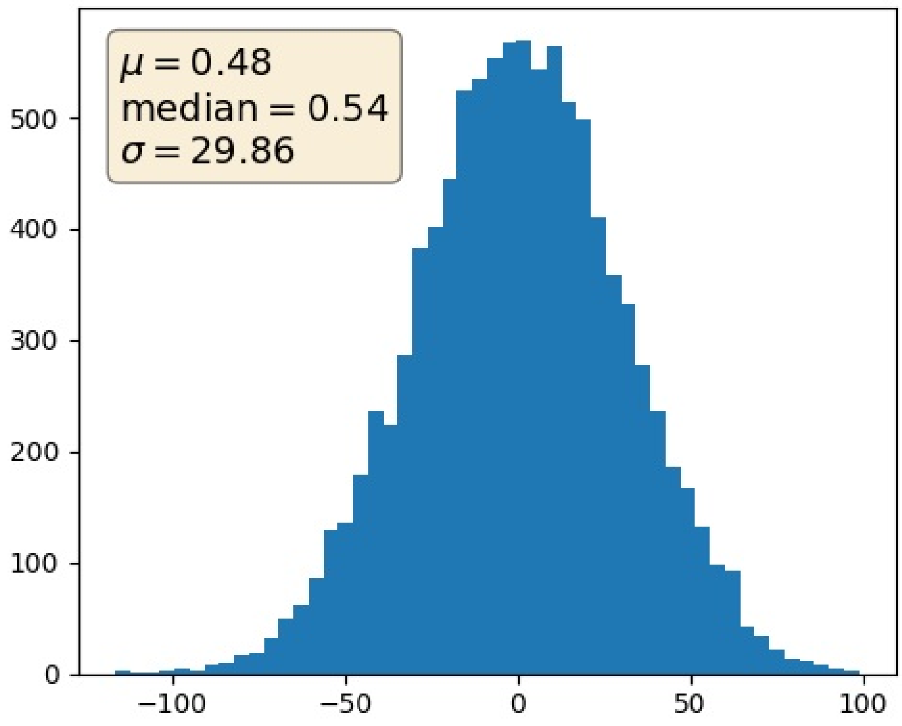
放置文本框

当使用文本框装饰轴时，两个有用的技巧是将文本放置在轴域坐标中（请参见变换教程），因此文本不会随着 x 或 y 轴的变化而移动。 你还可以使用文本的 `bbox` 属性，用 `Patch` 实例包围文本 - `bbox` 关键字参数接受字典，字典的键是补丁的属性。

```
np.random.seed(1234)
fig, ax = plt.subplots(1)
x = 30*np.random.randn(10000)
mu = x.mean()
median = np.median(x)
sigma = x.std()
textstr = '$\mu=%.2f$\n$\mathrm{median}=%.2f$\n$\sigma=%.2f$'%(m
u, median, sigma)

ax.hist(x, 50)
# these are matplotlib.patch.Patch properties
props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)

# place a text box in upper left in axes coords
ax.text(0.05, 0.95, textstr, transform=ax.transAxes, fontsize=14
,
        verticalalignment='top', bbox=props)
```


μ = 0.48
median = 0.54
σ = 29.86
500
400
300
200
100
0
−100
−50
0
50
100

术语表

英文	中文
Annotation	标注
Artist	艺术家
Axes	轴域
Axis	轴/坐标轴
Bézier	贝塞尔
Coordinate	坐标
Coordinate System	坐标系
Figure	图形
Handle	句柄
Handler	处理器
Image	图像
Legend	图例
Line	线条
Patch	补丁
Path	路径
Pick	拾取
Subplot	子图
Text	文本
Tick	刻度
Tick Label	刻度标签
Transformation	变换